# Configuration Manual

National

College of Ireland

MSc Research Project Data Analytics

## Sankara Subramanian Venkatraman Student ID: x18179541

School of Computing National College of Ireland

Supervisor: Mr. Hicham Rifai

#### National College of Ireland Project Submission Sheet School of Computing



Student Name:	Sankara Subramanian Venkatraman
Student ID:	x18179541
Programme:	Data Analytics
Year:	2020
Module:	MSc Research Project
Supervisor:	Mr. Hicham Rifai
Submission Due Date:	17/08/2020
Project Title:	Configuration Manual
Word Count:	1279
Page Count:	15

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

<u>ALL</u> internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	16th August 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<br/>each project (including multiple copies).□You must ensure that you retain a HARD COPY of the project, both for<br/>or□

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

## Sankara Subramanian Venkatraman x18179541

#### 1 Introduction

This configuration manual instructs the user to replicate the research project "Big Datadriven Performance Improvement of Traffic Flow Prediction and Speed Limit Classification using Deep Learning". It represents the storage, databases, software and hardware requirements, programming languages, and system setup used in the implementation of the research.

#### 2 System Configuration

Data sourced from the United Kingdom website <sup>1</sup> is huge and difficult to processes in the local system, so, the cloud system is preferred. Out of different cloud platforms Google Cloud Platform (GCP) *Google Cloud Platform* (GCP) *documentation* (n.d.) is chosen. It also provides free promotional credits of 276.60 euros for students shown in Figure 1.



Figure 1: Promotional Credits in GCP

#### 2.1 Storage

The raw data sourced from the United Kingdom is stored in the Google Cloud Storage (GCS) of Google Cloud Platform <sup>2</sup>. The road safety data from the year 2010 to 2018 for

<sup>1</sup> https://data.gov.uk/dataset/

 $<sup>^2</sup>$  https://cloud.google.com/storage/docs/creating-buckets

the United Kingdom is stored in the folder **accidents** and traffic flow data is stored in **traffic\_flow** folder. After data pre-processing, the data is stored in **cleaned\_data** folder.

=	Google Cloud Platform	Interfectorypodei	on 🗸		a, Search produc	ts and resources		<b>*</b>		2 0 O	: 🏟
	Storage	← Bucket detai	ls								\$LEARN
	Browser	traffic-flow-pres	diction								
-	Monitoring	OBJECTS CONF	GURATION	PERMISSIONS RI	TENTION LIFE	CYCLE					
=	Transfer										
۵	Transfer for on-premises	Buckets > traffic-flow	-prediction								
-10	Transfer Appliance	UPLOAD FILES UP	OAD FOLDER	CREATE FOLDER MM	NAGE HOLDS DE	LETE				Column disp	ilay options
•	Settings	Titler by object or fo	ider name prefi								۲
		None None	Size	Type	Created time 0	Storage class	Last modified	Public access 🔕	Encryption 🚱	Feteration expiry date 4	B Hold
		<ul> <li>Im accidents</li> </ul>		Polder							- 8
		Im cleaned.c		Folder							- 8
		D Dootgrase	910.9105	application/octet-stream	20 Jul 2020, 2	Standard	20 Jul 202	Not public	Google-managed key		Non
		🗋 🖿 taffe_fo		Folder							- 8

Figure 2: Google Cloud Storage Bucket

#### 2.2 Hardware

This step can be configured using Navigation Menu —> AI platform —> Notebook —> New Instance —> Smart Analytics Framework —> DataProc Hub [BETA] Figure 3a<sup>3</sup>. The required machine type, memory, CPU cores, GPU type and storage disk can be configured according to the requirements. The region of the compute engine should be in the EU according to GDPR. The instance provisioned in GCP compute engine has the below system configuration shown in the Figure 3b Data Engineering with Google Cloud Professional Certificate (n.d.).



(b) System Configuration

Figure 3: Hardware and System configuration

 $<sup>{}^{3}</sup>_{\tt https://cloud.google.com/ai-platform/notebooks/docs/create-new#before_you_begin}$ 

#### 2.3 Software

The software required for the analysis can also be configured using the option available in AI Notebook of GCP. Once the instance is provisioned, Jupyter Notebook with PySpark, Python3 and shell kernels are chosen using DataProc cluster's configuration option <sup>4</sup>. Navigation Menu —> AI platform —> Notebook —> OPEN JUPYTERLAB (Instance name) —> Cluster's configuration. The DataProc cluster is launched in a separate instance with Python3, PySpark and shell kernels software installed by default. Single-node-cluster is provisioned for this research. Finally, the Jupyter notebook with all the software components required for data cleaning, transformation, feature extraction and data mining models are obtained and shown in Figure 4.



Figure 4: Software requirements

#### 2.4 Database

After the data cleaning and transformation process using SparkSQL, the data is stored in the PostgreSQL database for Exploratory Data Analysis (EDA) and performing the final analysis using deep learning models. It can also provisioned in GCP <sup>5</sup> using the option **SQL** —> **CREATE INSTANCE** —> **PostgreSQL** (Figure 6). The instance name, password, location, region of data storage and database version can be chosen according to the requirements. The instance created for the research is shown in Figure 5.



Figure 5: Project Instance

<sup>&</sup>lt;sup>4</sup>https://cloud.google.com/dataproc/docs/concepts/components/jupyter

<sup>&</sup>lt;sup>5</sup>https://cloud.google.com/sql/docs/postgres/

	Google Clo	oud Plat	form	traffic-flow-pred	liction 🔻
3	SQL	÷	Crea	ite a PostgreSQL	. instanc
nsta	ance info				
nstan	ce ID				
hoice	e is permanent. Use	lowercase let	tters, numt	pers and hyphens. Start wit	h a letter.
etau	t user password	etarae' usar i	A pacewor	d is required for the user to	log in
etau etap earn	t user password password for the 'po more	stgres' user. /	A passwor	d is required for the user to	Generate
Locati Tor be Regior Choice	It user password password for the 'po more on @ tter performance, ke 1 2 is permanent	stgres' user. <i>i</i>	A passwor close to tł Zoj Cai	d is required for the user to the services that need it. ne be changed at any time	log in.
ocati or be egior hoice	It user password assword for the 'po more on @ tter performance, ke b is permanent entral1 (lowa)	stgres' user. /	A passwor close to th Zoi Cai	d is required for the user to	log in. Generate
ocati ocati or be tegior hoice	t user password assword for the 'po more on @ tter performance, ke is permanent e is permanent entral1 (lowa)	stgres' user. / ep your data	A passwor close to th Zoi Cai	d is required for the user to the services that need it. ne he changed at any time ny	log in. → Generate
Locati For be Region Choice Us-c	t user password assword for the 'po more on @ tter performance, ke is permanent entral1 (lowa) ase version	stgres' user. <i>i</i>	A passwor close to th Zoi Cai	d is required for the user to the services that need it. ne changed at any time ny	log in. Generate

Figure 6: PostgreSQL in GCP

#### 3 Data Preparation and Pre-Processing

Initially, the data downloaded from the United Kingdom website was in zip format for traffic flow data and CSV format for road safety data. Then the data from the zip file is extracted manually and stored in CSV format. Both, the files are then moved to GCP storage as mentioned in Section 2.1. Using the Dataproc instance mentioned in Section 2.2 and 2.3 with PySpark, Python3 and shell kernels installed, data cleaning and transformation is performed. The first part of all 4 .ipynb files uploaded in the code artefacts is used for data cleaning and transformation. SparkSession is initiated to read the data from GCS as shown in Figure 7a.



Figure 7: SparkSession and TempTable

Once the data read from GCS, the data is stored in Temporary tables (register-TempTable), for performing data cleaning and transformation using spark.sql is shown in Figure 7b. In the next step, the data is stored back to the cleaned\_data folder. As the research is carried as 4 experiments, the final tables required for all the experiments is created in PostgreSQL in prior using Data Definition Language (DDL) function of SQL shown in Figure 9a. PostgreSQL\_DDL\_Tables.sql available in the code artefacts is used to create tables in the PostgreSQL database. Finally, the cleaned transformed data stored in the GCS is uploaded from the Web UI using the option SQL -> Overview -> Import -> Source (GCS bucket), Destination (traffic-flow-prediction), Table (Table Name) as shown in Figure 9b.



(a) Postgres Tables

(b) Data Load in PostgreSQL

Figure 8: PostgreSQL Tables and Data Load

#### 4 Exploratory Data Analysis

The data stored in PostgreSQL is read through create\_engine of **sqlalchemy** and **Psy-copg2** package of Python3 shown in Figure 9 and stored in pandas dataframe. The second part of .ipynb files is used for Exploratory Data Analysis (EDA). It is carried using the stats function of **scipy** module, ADF test is conducted using the module **stats-models.tsa.stattools** and analysis are visualized using ticker, pyplot of **matplotlib** and **seaborn** modules. The below codes are designed using Python language only.

(1) Form sqlalcheny import create_engine	
postgres_str ('postgressl://lusermess):gossdr/gitaddress):gort//(domame)' .forms(usermane=MSOTGRES_passWolk, password=MSOTGRES_passWolk, ipaddress=MSOTGRES_passWolk, port=POSTGRES_passWolk, domame=VSOTGRES_passWolk, if	
<pre>[6]: # Create the connection as ns cnx = create_engine(postgres_tr) sns.set_style('white')</pre>	
<pre>[8]: df = pd.read_sql_query(''SELECT * FROM traffic_flow_2010_2018;'', cmx) print('Kurtosis of normal distribution: ()'.format(stats.k print('Kurtosis of normal distribution: ()'.format(stats.k)</pre>	urtosis(df.traffic_volume)))
(a) of ('court_date') = pd.to_datetime(df('court_date')) df('traffic_volume') = df('traffic_volume').estype(int) df('traffic_volume') = df('traffic_volume').estype(int) df('traffic_volume') = df('traffic_volume').estype(int)	



(b) Exploratory Data Analysis



#### 4.1 Experiment-1 Statistical Analysis Results

Figure 10 represents the code for skewness, kurtosis, traffic volume distribution, probability plot, seasonality from the year 2010 to 2018 without non-traffic parameters.



Figure 10: Exploratory Data Analysis Exp-1

#### 4.2 Experiment-2 Statistical Analysis Results

Figure 11 represents the code for skewness, kurtosis, traffic volume distribution, probability plot, seasonality from the year 2010 to 2018 with non-traffic parameters of weather, light and road surface conditions. Initially, the skewness value is out of range, due to outliers, then it has been removed and brought into the range of -10 to 10.



(a) Skewness and Kurtosis before removing outliers



(b) Skewness and Kurtosis after removing outliers



(d) Seasonality Traffic Volume with non-traffic parameters 2010-2018

Figure 11: Exploratory Data Analysis Exp-2



(c) Traffic Distribution with non-traffic parameters

## 4.3 Experiment-3 Statistical Analysis Results

Figure 12 represents the code for skewness, kurtosis and seasonality of traffic speed from the year 2010 to 2018 without non-traffic parameters.



(a) Skewness and Kurtosis



(b) Seasonality of Speed Limit without non-traffic parameters 2010-2018

Figure 12: Exploratory Data Analysis Exp-3

#### 4.4 Experiment-4 Statistical Analysis Results

Figure 13 represents the code for skewness, kurtosis and seasonality of traffic speed from the year 2010 to 2018 with non-traffic parameters of weather, light and road surface conditions.



(a) Skewness and Kurtosis



(b) Seasonality of Speed Limit with non-traffic parameters 2010-2018

Figure 13: Exploratory Data Analysis Exp-4

## 5 Feature Selection

After, data stationary check using the dickey-fuller test, the data is pivoted using pandas **pd.pivot\_table** and aggregate the traffic flow based on "year" from 2010 to 2018. After pivoting the data, MinMaxScaler normalization is applied from **sklearn.preprocessing** to normalize the data. The data is split into train and test using train\_test\_split of **sklearn.model\_selection**. Finally, using reshape of **numpy**, the data is reshaped from (data, features) to (training data, time steps, features) and (testing data, time steps, features). The below code snippets show the pivoting, normalization, train-test split and reshape feature incorporated in the analysis.

#### 5.1 Experiment-1 Feature Selection

Traffic-only parameter of traffic volume is used in the experiment.

i Impert mutgy Impert mutghollb.pyplot as plt Impert moth Impert Kerna from bioss.lppri Impert Sequential from bioss.lppri Impert Sequential from kissan.pypri Impert MindaScaler from kissan.matrici Impert mun_squardgeron from tislam.matrici Impert mun_squardgeron from tenss.lpyri Impert Dropout from kersa.lpyri Impert Dropout	

	# normalize the detaset scalar = MinMusScaler(feature_range=(0, 1)) dataset = scalar.fit_transform(dataset)
	<pre># split into train and test sets train = dataset[;,0:-1] print(train.shape) test = dataset[:,13] print(test.shape)</pre>
	(245730, 13) (245730,)
	<pre>from sklearn.model_selection import train_test_split trainX, testX, trainY, testY = train_test_split(train,test, test_size = 0.10, random_state = 42)</pre>
[124]:	<pre># reshape input to be [samples, time steps, features] trainK = numpy.reshape(trainK, (trainK.shape[0], 1, trainK.shape[1])) testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))</pre>
[125]:	<pre>print(trainX.shape, testX.shape,trainY.shape, testY.shape)</pre>
	(221157, 1, 13) (24573, 1, 13) (221157,) (24573,)

(b) Data Normalization

Figure 14: Feature Selection Exp-1

#### 5.2 Experiment-2 Feature Selection

Traffic volume and non-traffic parameters of weather, light and road surface conditions are used in this experiment.

		0	'travel_di 'peak_no n =['local_autho 'travel_di 'peak_non_p	rection' n_peak_ho inty_ons_ rection' eak_hours	<pre>'road_categor uurs', 'month') code','local_s 'road_categor ', 'month','co</pre>	<pre>y', 'noad_type  , hows'outen'] nuthority_id',' y', 'noad_type nunt_date'], ho</pre>	', )), traffic_c region_id','! ', wm'outer')	data_year_ Link_lengt	wise, h_kw',		
6]: <b>F</b>	inal_data										
	local_authoirty_ons_code	count_date	local_authority_id	region_id	link_length_km	travel_direction	road_category	road_type	peak_non_peak_ho	ars mont	h
	0 E06000001	2010-03-15	132	11	0.0000	1	5	2		2	3
	2 E05000001	2010-04-22	132	11	6.6000	3	3	1		1	2
	3 ED6000001	2010-04-22	132	11	6,6000	4	3	1		1	2
	4 E06000001	2010-06-15	132	11	0.0000	3	5	2		1	į
		e.varues									
i0]: 11-	<pre>dataset = dataset.a # normalize the dat scaler = MinMaxScal dataset = scaler.fi # solit into train</pre>	astype(' taset ler(feat it_trans	float32') ure_range=(0 form(dataset	9, 1)) :)							
0]: 1]:	<pre>dataset = dataset. # normalize the dat scaler = MinNaxScal dataset = scaler.fi # split into train train = dataset[;,4 print(train.shape) test = dataset[:,44 print(test.shape)</pre>	astype(' taset ler(feat it_trans and tes 0:-1]	float32') ure_range=(0 form(dataset t sets	9, 1)) :)							
8]:	<pre>dataset = dataset. # normalize the dat scaler = MinMaxScal dataset = scaler.fit # split into train train = dataset[:,4 print(train.shape) test = dataset[:,44 print(test.shap) (116561,40)</pre>	astype(' taset ler(feat it_trans and tes 0:-1]	float32') ure_range=(0 form(dataset t sets	9, 1)) ;)							
0]: 1]: 3]:	<pre>dataset = dataset. # normalize the dua scaler = MinMaxScal dataset = scaler.fl # split into train train = dataset[; 4 print(train.shape) test = dataset[; 4 print(test.shape) (116561, 49) (116561,) from sklearn.model trainX, testX, trai</pre>	<pre>sastype(' taset ler(feat ler(feat it_trans and tes ard tes ard tes ard it_ites</pre>	<pre>float32') ure_range=(0 form(dataset t sets on import tr tY = train_t</pre>	eain_terest_sp	st_split lit(train,1	:est, test_	size = 0.3	10, ran	dom_state = 4	12)	
50]: 51]: 53 <b>]:</b>	<pre>dataset = dataset.# normalize the ddt scalar = NinNedsCal dataset = scaler.fi # split into troin totaset[:,4 print(test.shap) (i16551,) from sklearn.model trainX, testX, trai # reshape input to trainX = nupy.resh</pre>	astype(' taset ler(feat it_trans and tes 0:-1] 0] 	<pre>float32') ure_range=(0 form(dataset t sets un import tr tY = train_t ples, time s ples, tix, (trainXx, (testX.sh</pre>	<pre>eain_te: est_sp: (.shape[0]</pre>	st_split lit(train,1 features] [0], 1, tri , 1, testX.	test, test_ binX.shape[ shape[1]))	size = 0.; 1]))	10, ran	dom_state = 4	12)	

(b) Data Normalization

Figure 15: Feature Selection Exp-2

#### **Experiment-3 Feature Selection** 5.3

The traffic speed limit is classified into 2 categories of Low Speed (1) and High Speed (2). Using to\_categorical of **Keras.utils.np\_utils**, the target column is converted to a categorical value. As the data is not a continuous value, normalization is not required for the classification problem.

[107]:	# Converting traffic-volume to time-series based data import pandas as pd
	<pre>import numpy as np speed_limit_data_year_wise = pd.pivot_table(df, valuess'speed_limit', indexs['local_authority_ons_code', 'count_date' ,'local_authority_id', 'n</pre>
	columns: 'year',aggfuncenp.median, fill_values0).rename_axis(None, axis:1).reset_index()
[100]:	speed_inst_owts_year_wise(zoid).unique() arrau([20, 0, 45, 35, 20, 50, 40, 50, 70, 25, 55, 55])
	speedinit(datpercinit(dat_percinit_(dat_percinit_(dat_percinit_(dat_percinit(dat_percinit(dat_percinit(dat_percinit(dat_percinit_(dat_percinit_(dat_percinit_(dat_percinit_(dat_percinit_(dat_percinit_(dat_percinit_perciniti_(dat_percinit_(dat_percinit_(dat_percinit_(d
	(a) Pivot Table Structure
[115]:	<pre>x = speed_limit_data_year_wise.iloc[:,5:-1]</pre>
[116]:	<pre>y = speed_limit_data_year_wise.iloc[:,-1]</pre>
	frem kerss.utils import to_categorical frem kerss.utils.mputils import to_categorical y = to_categorical(y)
[118]:	y, shape
[118]:	(45374, 3)
[119]:	<pre>x = np.array(x[:])</pre>
[120]:	<pre>from sklearn.model_selection import train_test_split x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.1,random_state = 42)</pre>
	<pre>x_train = x_train.reshape(x_train.shape[0],x_train.shape[1], 1) x_test = x_test.reshape(x_test.shape[0],x_test.shape[1], 1)</pre>
[122]:	<pre>print(x_train.shape) print(x_test.shape)</pre>
	(40836, 13, 1) (4538, 13, 1)
	<pre>print(y_train.shape) print(y_test.shape)</pre>
	(40836, 3) (4538, 3)
	(b) Data Reshape

Figure 16: Feature Selection Exp-3

#### 5.4**Experiment-4** Feature Selection

The traffic speed limit is classified into 2 categories of Low Speed (1) and High Speed (2) along with the non-traffic parameter of weather, light and road surface conditions. The target column is converted to a categorical value.





(b) Data Reshape

Figure 17: Feature Selection Exp-4

### 6 Code used for Deep Learning Models

Using Keras and TensorFlow package of Python3, the implementation of deep learning models is carried out. Also, hyper-parameters of epochs, batch size and train-test split ratio are changed and tested *Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization* (n.d.).

#### 6.1 Experiment-1 LSTM Model Traffic-only parameters

Different LSTM models of vanilla-LSTM, stacked-LSTM and Bi-directional LSTM models are applied to the trained dataset, and models are predicted and evaluated.

# create and fit the 151	TM network		
a creace and free the con			
Trom Keras.layers import	Bidirectional		
model = Sequential()			
<pre>#model.add(LSTM(100, act</pre>	tivation='softmax', return	sequences=True,	input shape=(1, 13))) #stacked-LSTM
	the stand and stand the stand	-(1 12))) # 1/	211- ICTM
model.add(LSIM(100, act)	lvation= reiu , input_snape	=(1, 13)) # Van'	illa- LSIM
#model.add(Bidirectional	(ISTM(50, activation='rel	('), input shape=	<pre>(1, 13))) #Bi-directional-LSTM</pre>
modebrada(brach ceetonae		()) enpuc_snapc=	(1) 15/// #50 00/0000000 25///
<pre>#model.add(LSTM(50, acti</pre>	ivation='relu'))		
model add(Dnonout(0.5))			
moder.aud(bropodc(0.3))			
<pre>model.add(Dense(1))</pre>			
model compile(loss='mae'	ontimizon ladam! motnic	c=[lacounacul_li	man (1)
model.compile(1033= mae	, opcimizer = adam , meeric	s=[ accuracy , i	mae j/
model.summary()			
Model: "sequential_5"			
· · ·			
laver (type)	Output Shape	Param #	
lstm 6 (LSTM)	(Nana 100)	45600	
150000 (1310)	(NOTE, 100)	43000	
	(1) (00)		
decourt 5 (Upopour)	(None, 100)	0	
diopode_5 (biopode)			
(bi opour_)			
dense_5 (Dense)	(None, 1)	101	
dense_5 (Dense)	(None, 1)	101	
dense_5 (Dense)  Total params: 45,701 Trainable params: 45,701 Non-trainable params: 0	(None, 1)	101	
dense_5 (Dense)  Total params: 45,701 Trainable params: 45,701 Non-trainable params: 0	(None, 1)	101	
dense_5 (Dense) 	(None, 1) (a) LSTM M	odel Sum	mary
dense_5 (Dense)  Total params: 45,701 Trainable params: 45,701 Non-trainable params: 0	(None, 1)	odel Sum	mary
dense_5 (Dense) 	(None, 1) (a) LSTM M	101 odel Sum	mary
dense_5 (Dense) 	(None, 1)	101 odel Sum testX, testY),validation. verboseit, shuffles <b>fals</b> e)	mary
dense_5 (Dense) 	(None, 1) (a) LSTM M VS58, bstch.size58, validation_data( ing(monitor'val_loss', patience10)),	101 odel Sum text, terty),validation. verbosei, shufflefalae)	mary
dense_5 (Dense) 	(None, 1) (a) LSTM M vs=50, batch_size=50, validation_data=( ing(enitor='val_less', patience=10)],	101 odel Sum test, testy),validation, verboses1, shuffle=False)	mary
dense_5 (Dense) 	(None, 1) (a) LSTM M hss50, bstch_sizer50, validation_date( ing(conitor='val.loss', patience:10)], -1 - 112 Jm/step - loss: 0.0000 - accu	101 odel Sum testx, testy),validation, verbose1, shuffle:False) ravy: 0.2249 - mac: 0.005	<b>mary</b> 39]it=0.30, 3 - yul 1051: 0.0074 - yul accuracy: 0.0264 - yul mat: 0.
<pre>dense_5 (Dense) dense_5 (Dense) Total params: 45,701 Trainable params: 45,701 Non-trainable params: 0 story = model.fit(trainx, train*, epod callbacks[EarlyStop ch J/50 N/351 [cmmmodel]</pre>	(None, 1) (a) LSTM M hss9, batch_sizes6, validation_data ing(monitor='val_loss', pathence10)], -] - 11s Jms/step - loss: 0.0003 - accu	101 odel Sum testX, testY),validation, verbose1, shufflesfalæe) racy: 0.9249 - mae: 0.008	<b>mary</b> split:0.10, 3 - val_loss: 0.0074 - val_sccuracy: 0.9264 - val_mas: 0.
dense_5 (Dense) dense_5 (Dense) Total params: 45,701 Trainable params: 45,701 Non-trainable params: 0 story = model.fit(trainx, trainx, epoc callbacks[farlyStopp ch 1/50 M1/391 [10]	(None, 1) (a) LSTM M hs580, batch_sizes50, validation_detac ping(conitor='val.loss', patience10)], 1) - 11s Jan/step - loss: 0.0003 - accu	101 odel Sum text, terY),validation, verbosel, shuffleffate) racy: 0.9249 - mac: 0.008	mary split=0.10, 1 - val_tors: 0.0074 - val_accuracy: 0.9264 - val_mae: 0. 8 - val_tors: 0.0054 - val_accuracy: 0.9264 - val_mae.0
dense_5 (Dense) 	(None, 1) (a) LSTM M hs58, bstch_size58, validation_data( ping(monitors'val_loss', patience10)], 1) - 115 3ms/step - loss: 0.0003 - accu -1) - 105 3ms/step - loss: 0.0008 - accu	101 odel Sum textX, textY),validation, verbose1, shuffle:False) racy: 0.5249 - mae: 0.008 racy: 0.5249 - mae: 0.008	<b>mary</b> splite=0.0, 13 - val_loss: 0.0074 - val_eccuracy: 0.9264 - val_mas: 0. 8 - val_loss: 0.0054 - val_eccuracy: 0.9264 - val_mas: 0.
dense_5 (Dense) dense_5 (Dense) Total params: 45,701 Trainable params: 45,701 Non-trainable params: 0 story = model.fit(trainX, trainY, epod callbacks=[terlyStop th/3901[	(None, 1) (None, 1) (a) LSTM M hss9, batch_sizes5, validation_datac ping(contact val.less', patiences10)], .1 - 11s Jms/step - loss: 0.0003 - accu	101 odel Sum testx, testy),validation racy: 0.9249 - mae: 0.008 racy: 0.9249 - mae: 0.008	.mary split=0.10, 13 - val_loss: 0.0074 - val_accuracy: 0.5264 - val_mas: 0. 4 - val_loss: 0.0545 - val_accuracy: 0.5264 - val_mas: 0. 5 - val_loss: 0.0555 - val_accuracy: 0.5264 - val_mas: 0.
dense_5 (Dense) dense_5 (Dense) Total params: 45,701 Trainable params: 45,701 Non-trainable params: 0 story = model.fit(trainx, trainy, epoc callbacks:[tarlyStor trainable params: 0 Non-trainable params: 0 Non-tr	(None, 1) (a) LSTM M (bs:50, bstch_size:50, validation_datac( pig(donitor='val.loss', patience:10)]; ·] - 11s 3ms/step - loss: 0.0003 - accu ·] - 10s 3ms/step - loss: 0.0005 - accu	101 odel Sum testy, testy),volidation, verboses1, shuffle=False) racy: 0.9249 - mae: 0.005 cacy: 0.9249 - mae: 0.005 cacy: 0.9249 - mae: 0.005	mary splite0.10, 3 - val_loss: 0.0074 - val_escuracy: 0.9264 - val_mas: 0. 8 - val_oss: 0.0054 - val_ecuracy: 0.9264 - val_mas: 0. 3 - val_oss: 0.0056 - val_ecuracy: 0.9264 - val_mas: 0.
dense_5 (Dense) 	(None, 1) (None, 1) (a) LSTM M hs:50, bitch_lize50, validation_deta( ping(montor='val_bes', patience:10)], 1) - 115 Jms/step - loss: 0.0003 - accu -] - 108 Jms/step - loss: 0.0005 - accu	101 odel Sum testX, testY),validation, verbosec1, shuffle=false) racy: 0.9249 - mae: 0.006 racy: 0.9249 - mae: 0.006	mary splite0.10, 1 - val_loss: 0.0074 - val_sccuracy: 0.5264 - val_mes: 0. 8 - val_loss: 0.0054 - val_sccuracy: 0.5264 - val_mes: 0. 3 - val_loss: 0.055 - val_eccuracy: 0.5264 - val_mes: 0.
dense_5 (Dense) dense_5 (Dense) Total params: 45,701 Trainable params: 45,701 Non-trainable params: 0 story = model.fit(trainX, trainY, equal callbacks=[farlyStopp ch 1/50 H/501 H/501 (Job 1/50 H/501 H/501 (Job 1/50 H/501 H/5	(None, 1) (None, 1) (a) LSTTM M hs50, batch_size=50, validation_datace ig(contor='val_less', patience10)]. 1) - 115 Jms/step - loss: 0.0083 - accu 1) - 105 Jms/step - loss: 0.0085 - accu 1) - 105 Jms/step - loss: 0.0085 - accu	101 odel Sum testy, testy),volidation, verbosel, shuffle=False) racy: 0.9249 - mae: 0.008 racy: 0.9249 - mae: 0.008 cacy: 0.9249 - mae: 0.005	split=0.10,           3 - val_loss: 0.0074 - val_accuracy: 0.9264 - val_mas: 0.           4 - val_loss: 0.0054 - val_accuracy: 0.9264 - val_mas: 0.           3 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mas: 0.           3 - val_loss: 0.0054 - val_accuracy: 0.9264 - val_mas: 0.           3 - val_loss: 0.0054 - val_accuracy: 0.9264 - val_mas: 0.
dense_5 (Dense) dense_5 (Dense) Total params: 45,701 Trainable params: 45,701 Non-trainable params: 0 story = model.fit(trainX, trainY, epod callbacks[EarLyStop th J509 1/5091 [	(None, 1) (None, 1) (a) LSTM M hs58, batch_size58, validation_data( ping(mottor='val_loss', patterce:10)], a) - 115 3ms/step - loss: 0.0003 - accu -1) - 105 3ms/step - loss: 0.0005 - accu -1) - 108 3ms/step - loss: 0.0007 - accu	101 odel Sum test, testy),validation verbosei, shufflesfalæ) racy: 0.9249 - mae: 0.006 racy: 0.9249 - mae: 0.006 racy: 0.9249 - mae: 0.006	mary splite0.10, 13 - vallos: 0.0074 - val_eccuracy: 0.5264 - val_mas: 0. 8 - val_os: 0.0054 - val_eccuracy: 0.5264 - val_mas: 0. 3 - val_os: 0.0055 - val_eccuracy: 0.5264 - val_mas: 0. 2 - val_os: 0.0054 - val_eccuracy: 0.5264 - val_mas: 0.
<pre>dence_5 (Dence) dence_5 (Dence) Total params: 45,701 Trainable params: 45,701 Trainable params: 0 story = model.fit(trainX, trainY, epod callbacks=[terlyStop th/3951] th/3951 th/3951 th/3951</pre>	(None, 1) (None, 1) (a) LSTM M hss9, batch_sizes5, validation_data( ping(contine="validation", pathement()], -1 - 11s Jan/step - loss: 0.0003 - accu -1 - 108 Jan/step - loss: 0.0002 - accu	101 odel Sum testx, testy),validation racy: 0.9249 - mae: 0.008 racy: 0.9249 - mae: 0.006 racy: 0.9249 - mae: 0.006 racy: 0.9249 - mae: 0.006 racy: 0.9249 - mae: 0.006 racy: 0.9249 - mae: 0.006	split=0.10,           1 - val_loss: 0.0074 - val_accuracy: 0.9264 - val_mas: 0.           3 - val_loss: 0.0054 - val_accuracy: 0.9264 - val_mas: 0.           3 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mas: 0.           2 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mas: 0.           2 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mas: 0.           2 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mas: 0.
dense_5 (Dense) dense_5 (Dense) 	(None, 1) (None, 1) (a) LSTM M hs580, bstch_size50, validation_datae( pig(fonitor='val.loss', vatience:10)]; *) - 11s 3ms/step - loss: 0.0083 - accu -) - 10s 3ms/step - loss: 0.0083 - accu -) - 10s 3ms/step - loss: 0.0062 - accu	101 odel Sum testy, testy),volidation, verboses1, shuffle=Falee) racy: 0.9249 - mae: 0.006 racy: 0.9249 - mae: 0.006 racy: 0.9249 - mae: 0.006 racy: 0.9249 - mae: 0.006	mary pplite0.0, 3 - val_loss: 0.0074 - val_escuracy: 0.9264 - val_mas: 0. 4 - val_loss: 0.0054 - val_ecuracy: 0.9264 - val_mas: 0. 3 - val_loss: 0.0054 - val_ecuracy: 0.9264 - val_mas: 0. 2 - val_loss: 0.0050 - val_ecuracy: 0.9264 - val_mas: 0.
dense_5 (Dense) 	(None, 1) (None, 1) (a) LSTM M hs:50, bitch_size50, validation_data( ing(montor='val_less', patience:10)], =! - 11s Jas/step - loss: 0.0003 - accu -! - 10s Jas/step - loss: 0.0005 - accu -! - 10s Jas/step - loss: 0.0002 - accu :! - 10s Jas/step - loss: 0.0002 - accu :! - 10s Jas/step - loss: 0.0002 - accu	101 odel Sum textX, textY),validation, verbose1, shuffle=False) racy: 0.5249 - mae: 0.005 racy: 0.5249 - mae: 0.005 racy: 0.5249 - mae: 0.005 racy: 0.5249 - mae: 0.005 racy: 0.5249 - mae: 0.005	mary splite0.10, 1 - val_loss: 0.0074 - val_eccuracy: 0.9264 - val_mas: 0. 8 - val_loss: 0.0054 - val_eccuracy: 0.9264 - val_mas: 0. 3 - val_loss: 0.0054 - val_eccuracy: 0.9264 - val_mas: 0. 2 - val_loss: 0.0059 - val_eccuracy: 0.9264 - val_mas: 0. 2 - val_loss: 0.0059 - val_eccuracy: 0.9264 - val_mas: 0. 2 - val_loss: 0.0059 - val_eccuracy: 0.9264 - val_mas: 0.
<pre>dence (or opency) dense_5 (Dense) </pre>	(None, 1) (None, 1) (a) LSTTM M hs50, batch_sizes50, validation_datac prigromitors'valless', patiences10)], 1 = 11s Jms/step - loss: 0.0005 - accu -1 = 10s Jms/step - loss: 0.0005 - accu	101 Odel Sum testy, testy),volidation, rerbosent, shufflesfalse) racy: 0.9249 - mae: 0.008 racy: 0.9249 - mae: 0.005 racy: 0.9249 - mae: 0.005 racy: 0.9249 - mae: 0.005 racy: 0.9249 - mae: 0.005	Split=0.10, split=0.10, 1 - val_loss: 0.0074 - val_accuracy: 0.9264 - val_mas: 0. 8 - val_loss: 0.0054 - val_accuracy: 0.9264 - val_mas: 0. 3 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mas: 0. 2 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mas: 0. 2 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mas: 0.
dense_5 (Dense) 	(None, 1) (None, 1) (a) LSTM M hs58, batch_size58, validation_data( ping(mottors'val_loss', pattercein)), 	101 odel Sum test, testy),validation, verbose1, shuffle=False) racy: 0.9249 - mai: 0.060 racy: 0.9249 - mai: 0.060 racy: 0.9249 - mai: 0.060 racy: 0.9249 - mai: 0.060 racy: 0.9249 - mai: 0.060	mary splite.0.10, 1 - val_loss: 0.0074 - val_accuracy: 0.9264 - val_mas: 0. 4 - val_loss: 0.0056 - val_accuracy: 0.9264 - val_mas: 0. 3 - val_loss: 0.0056 - val_accuracy: 0.9264 - val_mas: 0. 2 - val_loss: 0.0056 - val_accuracy: 0.9264 - val_mas: 0. 2 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mas: 0. 2 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mas: 0. 2 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mas: 0. 2 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mas: 0.
<pre>dence_5 (Dense) dense_5 (Dense) Total params: 45,701 Trainable params: 45,701 Non-trainable params: 6 story = model.fit(trainX, trainY, epod calbacks=[tarlyStop trainable params: 7 trainable params: 6 trainable params: 7 trainable params: 7 trai</pre>	(None, 1) (None, 1) (a) LSTM M hs:9, bitch_sizes5, validation_datas ping(contion='val_loss', patience:10)]; -1 - 11s Jan/step - loss: 0.0003 - accu -2 - 108 Jan/step - loss: 0.0002 - accu -1 - 108 Jan/step - loss: 0.0002 - accu	101 Odel Sum testx, testy),validation, racy: 0.9249 - mae: 0.008 racy: 0.9249 - mae: 0.006 racy: 0.9249 - mae: 0.006 racy: 0.9249 - mae: 0.006 racy: 0.9249 - mae: 0.006 racy: 0.9249 - mae: 0.006	splite0.10,           1 - val_loss: 0.0074 - val_accuracy: 0.9264 - val_mes: 0.           3 - val_loss: 0.0054 - val_accuracy: 0.9264 - val_mes: 0.           3 - val_loss: 0.0054 - val_accuracy: 0.9264 - val_mes: 0.           2 - val_loss: 0.0054 - val_accuracy: 0.9264 - val_mes: 0.           2 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mes: 0.           2 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mes: 0.           2 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mes: 0.           2 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mes: 0.
dense_5 (Dense) dense_5 (Dense) 	(None, 1) (None, 1) (a) LSTM M hs50, bstch_size50, validation_data=[ pig(fonitor='val.loss; , patience:10]] +] - 115 3ms/step - loss; 0.0003 - accu -] - 105 3ms/step - loss: 0.0005 - accu -] - 105 3ms/step - loss: 0.0002 - accu	101 odel Sum resty, testy),volidation, verboses1, shuffle=False) racy: 0.9249 - mae: 0.006 racy: 0.9249 - mae: 0.006	mary pplite0.10, 1 - val_loss: 0.0074 - val_sccuracy: 0.9264 - val_mas: 0. 8 - val_oss: 0.0094 - val_sccuracy: 0.9264 - val_mas: 0. 3 - val_oss: 0.0095 - val_sccuracy: 0.9264 - val_mas: 0. 2 - val_oss: 0.0095 - val_sccuracy: 0.9264 - val_mas: 0. 3 - val_oss: 0.0095 - val_sccuracy: 0.9264 - val_mas: 0. 1 - val_oss: 0.0095 - val_sccuracy: 0.9264 - val_mas: 0. 1 - val_oss: 0.0095 - val_sccuracy: 0.9264 - val_mas: 0.
dense_5 (Dense) 	(None, 1) (None, 1) (a) LSTM M hs:50, bitch_lize50, validation_detect ping(montor="val_best", patience:10)]; +1 - 11s Jms/step - loss: 0.0003 - accu +2 - 108 Jms/step - loss: 0.0005 - accu +1 - 108 Jms/step - loss: 0.0002 - accu +1 - 108 Jms/step - loss: 0.0001 - accu	101 Odel Sum testX, testY),validation, verboseil, shuffletfalæ) racy: 0.9249 - mae: 0.006 racy: 0.9249 - mae: 0.006	mary splite0.10, 1 - val_loss: 0.0074 - val_eccuracy: 0.9264 - val_mas: 0. 8 - val_loss: 0.0054 - val_eccuracy: 0.9264 - val_mas: 0. 3 - val_loss: 0.0054 - val_eccuracy: 0.9264 - val_mas: 0. 2 - val_loss: 0.0055 - val_eccuracy: 0.9264 - val_mas: 0. 2 - val_loss: 0.0055 - val_eccuracy: 0.9264 - val_mas: 0. 1 - val_loss: 0.0055 - val_eccuracy: 0.9264 - val_mas: 0. 1 - val_loss: 0.0055 - val_eccuracy: 0.9264 - val_mas: 0. 1 - val_loss: 0.0055 - val_eccuracy: 0.9264 - val_mas: 0.
dense_5 (Dense) 	(None, 1) (None, 1) (a) LSTTM M hss0, batch_sizes0, validation_datac prigromitors'valless', patiences10), 1 = 11s Jan/step - loss: 0.0003 - accu - 1 = 10s Jan/step - loss: 0.0005 - accu - 1 = 10s Jan/step - loss: 0.0005 - accu - 1 = 10s Jan/step - loss: 0.0002 - accu - 1 = 10s Jan/step - loss: 0.0002 - accu - 1 = 10s Jan/step - loss: 0.0002 - accu - 1 = 10s Jan/step - loss: 0.0002 - accu - 1 = 10s Jan/step - loss: 0.0002 - accu - 1 = 10s Jan/step - loss: 0.0002 - accu - 1 = 10s Jan/step - loss: 0.0002 - accu	101 odel Sum textX, terY),validation, verbose-1, shuffle=False) racy: 0.9249 - mac: 0.006 racy: 0.9249 - mac: 0.006	
dense_5 (Dense) dense_5 (Dense) Total params: 45,701 Trainable params: 45,701 Non-trainable params: 6 story = model.fit(trainX, trainY, epod callbacks[EarLyStop th/3951 [	(None, 1) (None, 1) (a) LSTM M hsss, hath_sizeSs, validation_datus( ping(montor='val_less', patience=10)]; -1 - 115 Jms/step - loss: 0.0003 - accu -2 - 106 Jms/step - loss: 0.0005 - accu -1 - 108 Jms/step - loss: 0.0002 - accu -1 - 108 Jms/step - loss: 0.0001 - accu	101 Odel Sum verbosel, shufflesfalee) racy: 0.9249 - mai: 0.005 racy: 0.9249 - mai: 0.005	splite.0.10,           3 - val_loss: 0.0074 - val_eccuracy: 0.5264 - val_mes: 0.           4 - val_oss: 0.0055 - val_eccuracy: 0.5264 - val_mes: 0.           5 - val_oss: 0.0055 - val_eccuracy: 0.5264 - val_mes: 0.           2 - val_oss: 0.0055 - val_eccuracy: 0.5264 - val_mes: 0.           2 - val_oss: 0.0055 - val_eccuracy: 0.5264 - val_mes: 0.           2 - val_oss: 0.0055 - val_eccuracy: 0.5264 - val_mes: 0.           2 - val_oss: 0.0055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_oss: 0.0055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_oss: 0.0055 - val_eccuracy: 0.5264 - val_mes: 0.
<pre>choper_s (chopers) dense_5 (Dense) </pre>	(None, 1) (None, 1) (a) LSTM M hss9, batch_sizes5, validation_datas ping(sonitor='val_loss', patiences10}]; +1 - 11s Jan/step - loss: 0.0003 - accu -2 - 108 Jan/step - loss: 0.0002 - accu -1 - 108 Jan/step - loss: 0.0001 - accu -1 - 108 Jan/step - loss: 0.0001 - accu	101 odel Sum testx, testy),validation, racy: 0.9249 - mai: 0.008 racy: 0.9249 - mai: 0.006 racy: 0.9249 - mai: 0.006	mary split+0.10, 3 - val_loss: 0.0074 - val_sccuracy: 0.9264 - val_mes: 0. 4 - val_loss: 0.0054 - val_sccuracy: 0.9264 - val_mes: 0. 3 - val_loss: 0.0054 - val_sccuracy: 0.9264 - val_mes: 0. 2 - val_loss: 0.0055 - val_sccuracy: 0.9264 - val_mes: 0. 2 - val_loss: 0.0055 - val_sccuracy: 0.9264 - val_mes: 0. 1 - val_loss: 0.0055 - val_sccuracy: 0.9264 - val_mes: 0. 1 - val_loss: 0.0055 - val_sccuracy: 0.9264 - val_mes: 0. 1 - val_loss: 0.0055 - val_sccuracy: 0.9264 - val_mes: 0. 1 - val_loss: 0.0055 - val_sccuracy: 0.9264 - val_mes: 0. 1 - val_loss: 0.0055 - val_sccuracy: 0.9264 - val_mes: 0. 1 - val_loss: 0.0059 - val_sccuracy: 0.9264 - val_mes: 0. 1 - val_loss: 0.0059 - val_sccuracy: 0.9264 - val_mes: 0.
dense_5 (Dense) 	(None, 1) (None, 1) (a) LSTM M hs58, batch_size58, validation_detac{ ping(monitors'val_less', pattence10)], 	101 odel Sum testy, testy),volidation, verboses1, shuffle=False) racy: 0.9249 - mae: 0.006 racy: 0.9249 - mae: 0.006	mary plite0.10, 3 - val_loss: 0.0074 - val_eccuracy: 0.9264 - val_mas: 0. 4 - val_oss: 0.0054 - val_eccuracy: 0.9264 - val_mas: 0. 3 - val_oss: 0.0055 - val_eccuracy: 0.9264 - val_mas: 0. 2 - val_oss: 0.0055 - val_eccuracy: 0.9264 - val_mas: 0. 2 - val_oss: 0.0055 - val_eccuracy: 0.9264 - val_mas: 0. 1 - val_oss: 0.0055 - val_eccuracy: 0.9264 - val_mas: 0. 1 - val_oss: 0.0055 - val_eccuracy: 0.9264 - val_mas: 0. 1 - val_oss: 0.0055 - val_eccuracy: 0.9264 - val_mas: 0. 1 - val_oss: 0.0055 - val_eccuracy: 0.9264 - val_mas: 0. 1 - val_oss: 0.0055 - val_eccuracy: 0.9264 - val_mas: 0. 1 - val_oss: 0.0050 - val_eccuracy: 0.9264 - val_mas: 0.
dense_5 (Dense) 	(None, 1) (None, 1) (a) LSTM M hs:9, bith,size50, validation_datas ping(sontor='val_loss', patience:10)]; -1 - 11s 3ms/step - loss: 0.0083 - accu -1 - 10s 3ms/step - loss: 0.0085 - accu -1 - 10s 3ms/step - loss: 0.0082 - accu -1 - 10s 3ms/step - loss: 0.0081 - accu -1 - 10s 3ms/step - loss: 0.0081 - accu -1 - 10s 3ms/step - loss: 0.0081 - accu	101 Odel Sum testX, testY),validation, verbosel, shuffletflet racy: 0.9249 - mae: 0.006 racy: 0.9249 - mae: 0.006	mary splite0.10, 1 - val_loss: 0.0074 - val_eccuracy: 0.9264 - val_mes: 0. 6 - val_oss: 0.0054 - val_eccuracy: 0.9264 - val_mes: 0. 7 - val_loss: 0.0054 - val_eccuracy: 0.9264 - val_mes: 0. 9 - val_loss: 0.0054 - val_eccuracy: 0.9264 - val_mes: 0. 1 - val_loss: 0.0055 - val_eccuracy: 0.9264 - val_mes: 0. 1 - val_loss: 0.0055 - val_eccuracy: 0.9264 - val_mes: 0. 1 - val_loss: 0.0055 - val_eccuracy: 0.9264 - val_mes: 0. 1 - val_loss: 0.0055 - val_eccuracy: 0.9264 - val_mes: 0. 1 - val_loss: 0.0055 - val_eccuracy: 0.9264 - val_mes: 0. 1 - val_loss: 0.0055 - val_eccuracy: 0.9264 - val_mes: 0. 1 - val_loss: 0.0055 - val_eccuracy: 0.9264 - val_mes: 0. 1 - val_loss: 0.0055 - val_eccuracy: 0.9264 - val_mes: 0. 1 - val_loss: 0.0055 - val_eccuracy: 0.9264 - val_mes: 0. 1 - val_loss: 0.0055 - val_eccuracy: 0.9264 - val_mes: 0. 1 - val_loss: 0.0055 - val_eccuracy: 0.9264 - val_mes: 0.
dense_5 (Dense) 	(None, 1) (None, 1) (a) LSTTM M hs50, batch_size50, validation_dates (b) LSTM A (c) LSTM A (c) LSTM A (c) LSSTM A (	101 Odel Sum testX, testY),validation, verboses1, shuffle=False) racy: 0.9249 - mae: 0.006 racy:	splite0.10,           1 - val_loss: 0.0074 - val_securacy: 0.9264 - val_mes: 0.           3 - val_loss: 0.0094 - val_securacy: 0.9264 - val_mes: 0.           4 - val_loss: 0.0095 - val_securacy: 0.9264 - val_mes: 0.           2 - val_loss: 0.0095 - val_securacy: 0.9264 - val_mes: 0.           2 - val_loss: 0.0095 - val_securacy: 0.9264 - val_mes: 0.           1 - val_loss: 0.0095 - val_securacy: 0.9264 - val_mes: 0.           1 - val_loss: 0.0095 - val_securacy: 0.9264 - val_mes: 0.           1 - val_loss: 0.0095 - val_securacy: 0.9264 - val_mes: 0.           1 - val_loss: 0.0095 - val_securacy: 0.9264 - val_mes: 0.           1 - val_loss: 0.0095 - val_securacy: 0.9264 - val_mes: 0.           1 - val_loss: 0.0095 - val_securacy: 0.9264 - val_mes: 0.           1 - val_loss: 0.0095 - val_securacy: 0.9264 - val_mes: 0.           1 - val_loss: 0.0095 - val_securacy: 0.9264 - val_mes: 0.           1 - val_loss: 0.0095 - val_securacy: 0.9264 - val_mes: 0.           1 - val_loss: 0.0095 - val_securacy: 0.9264 - val_mes: 0.           1 - val_loss: 0.0095 - val_securacy: 0.9264 - val_mes: 0.
dense_5 (Dense) 	(None, 1) (None, 1) (a) LSTM M hs:50, bitch_lize50, validation_detato ping(montors*val_less', patience:10)]; -1 - 11s Jms/step - loss: 0.0003 - accu -2 - 10s Jms/step - loss: 0.0005 - accu -1 - 10s Jms/step - loss: 0.0002 - accu -2 - 10s Jms/step - loss: 0.0002 - accu -1 - 10s Jms/step - loss: 0.0002 - accu -2 - 10s Jms/step - loss: 0.0002 - accu -1 - 10s Jms/step - loss: 0.0001 - accu -1 - 11s Jms/step - loss: 0.0001 - accu -1 - 11s Jms/step - loss: 0.0001 - accu	101 Odel Sum testx, testy),validation, verbosei, shufflesfalse) racy: 0.9249 - mas: 0.006 racy: 0.	splite.0.10,           1 - val_loss: 0.0074 - val_accuracy: 0.9264 - val_mes: 0.           3 - val_loss: 0.0074 - val_accuracy: 0.9264 - val_mes: 0.           3 - val_loss: 0.0056 - val_accuracy: 0.9264 - val_mes: 0.           2 - val_loss: 0.0056 - val_accuracy: 0.9264 - val_mes: 0.           2 - val_loss: 0.0056 - val_accuracy: 0.9264 - val_mes: 0.           2 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mes: 0.           2 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mes: 0.           1 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mes: 0.           1 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mes: 0.           1 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mes: 0.           1 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mes: 0.           1 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mes: 0.           1 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mes: 0.           1 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mes: 0.           1 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mes: 0.           1 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mes: 0.           1 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mes: 0.           1 - val_loss: 0.0055 - val_accuracy: 0.9264 - val_mes: 0.
dense_5 (Dense)	(None, 1) (None, 1) (a) LSTTM M (b) Statution (detail) (b) Statution (detail) (c) Statution (deta	101 Odel Sum test, terty, volidation, verbosent, shufflesfalse) racy: 0.9249 - mae: 0.008 racy: 0.9249 - mae: 0.008 racy: 0.9249 - mae: 0.006 racy: 0	mary splite0.10, splite0.10, d - val_loss: 0.00%4 - val_accuracy: 0.9264 - val_mes: 0. d - val_loss: 0.00%4 - val_accuracy: 0.9264 - val_mes: 0. d - val_loss: 0.00%5 - val_mecuracy: 0.9264 - val_mes: 0. d - val_loss: 0.00%5 - val_mecuracy: 0.9264 - val_mes: 0. d - val_loss: 0.00%5 - val_mecuracy: 0.9264 - val_mes: 0. d - val_loss: 0.00%5 - val_mecuracy: 0.9264 - val_mes: 0. d - val_loss: 0.00%5 - val_mecuracy: 0.9264 - val_mes: 0. d - val_loss: 0.00%5 - val_mecuracy: 0.9264 - val_mes: 0. d - val_loss: 0.00%5 - val_mecuracy: 0.9264 - val_mes: 0. d - val_loss: 0.00%5 - val_mecuracy: 0.9264 - val_mes: 0. d - val_loss: 0.00%5 - val_mecuracy: 0.9264 - val_mes: 0. d - val_loss: 0.00%5 - val_mecuracy: 0.9264 - val_mes: 0. d - val_loss: 0.00%5 - val_mecuracy: 0.9264 - val_mes: 0.
dense_5 (Dense) 	(None, 1) (None, 1) (a) LSTM M (b) LSTM M (c) LSTM A (c) LSTM A (c) LSTM A (c) LSSTM A (	101 odel Sum testX, testY),volidation, verboseil, shuffle=False) racy: 0.9249 - mae: 0.006 racy: 0.9249 - mae: 0.006	aplits0.10,           3 - val_loss: 0.6074 - val_eccuracy: 0.5264 - val_mes: 0.           4 - val_olss: 0.6054 - val_eccuracy: 0.5264 - val_mes: 0.           5 - val_olss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.           2 - val_loss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.           2 - val_olss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.           2 - val_olss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.
dense_5 (Dense) 	(None, 1) (None, 1) (a) LSTTM M hs:9, bith,size50, validation_data( ping(sontor='val_loss', patience:10)]; -1 - 11s Jms/step - loss: 0.0003 - accu -2 - 108 Jms/step - loss: 0.0005 - accu -1 - 108 Jms/step - loss: 0.0002 - accu -1 - 108 Jms/step - loss: 0.0001 - accu -1 - 118 Jms/step - loss: 0.0001 - accu -1 - 118 Jms/step - loss: 0.0001 - accu -1 - 118 Jms/step - loss: 0.0001 - accu	101 Odel Sum everbosei, shuffletfin, verbosei, shuffletfin, racy: 0.9249 - mae: 0.005 racy: 0.924	splite.0.10;           splite.0.10;           3 - val_loss: 0.0074 - val_eccuracy: 0.5264 - val_mes: 0.           6 - val_olss: 0.0054 - val_eccuracy: 0.5264 - val_mes: 0.           3 - val_loss: 0.0054 - val_eccuracy: 0.5264 - val_mes: 0.           2 - val_loss: 0.0050 - val_eccuracy: 0.5264 - val_mes: 0.           2 - val_loss: 0.0050 - val_eccuracy: 0.5264 - val_mes: 0.           2 - val_loss: 0.0050 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_loss: 0.0050 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_loss: 0.0055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_loss: 0.0055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_loss: 0.0055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_loss: 0.0055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_loss: 0.0055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_loss: 0.0056 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_loss: 0.0056 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_loss: 0.0056 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_loss: 0.0056 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_loss: 0.0051 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_loss: 0.0051 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_loss: 0.0051 - val_eccuracy: 0.5264 - val_mes: 0.
dense_5 (Dense)           dense_5 (Dense)	(None, 1) (None, 1) (a) LSTTM M hss0, batch_sizes0, validation_datac pigeonitor='validas:, patiences10}, 1 = 11s Jms/step - loss: 0.0003 - accu - 1 = 10s Jms/step - loss: 0.0005 - accu - 1 = 10s Jms/step - loss: 0.0005 - accu - 1 = 10s Jms/step - loss: 0.0005 - accu - 1 = 10s Jms/step - loss: 0.0005 - accu - 1 = 10s Jms/step - loss: 0.0005 - accu - 1 = 10s Jms/step - loss: 0.0005 - accu - 1 = 10s Jms/step - loss: 0.0005 - accu - 1 = 10s Jms/step - loss: 0.0005 - accu - 1 = 10s Jms/step - loss: 0.0005 - accu - 1 = 10s Jms/step - loss: 0.0005 - accu - 1 = 10s Jms/step - loss: 0.0005 - accu - 1 = 10s Jms/step - loss: 0.0005 - accu - 1 = 10s Jms/step - loss: 0.0005 - accu - 1 = 10s Jms/step - loss: 0.0005 - accu - 1 = 10s Jms/step - loss: 0.0005 - accu - 1 = 10s Jms/step - loss: 0.0005 - accu	101 Codel Summ testX, testY),velidation, verboses1, shuffle=False) racy: 0.9249 - mac: 0.006 racy: 0.9249 - mac: 0.006 racy	splite.0.10,           3-val_loss: 0.0074 - val_securacy: 0.9264 - val_mese 0.           4. val_loss: 0.0095 - val_securacy: 0.9264 - val_mese 0.           5. val_loss: 0.0095 - val_securacy: 0.9264 - val_mese 0.           2. val_loss: 0.0095 - val_securacy: 0.9264 - val_mese 0.           2. val_loss: 0.0095 - val_securacy: 0.9264 - val_mese 0.           2. val_loss: 0.0095 - val_securacy: 0.9264 - val_mese 0.           2. val_loss: 0.0095 - val_securacy: 0.9264 - val_mese 0.           1. val_loss: 0.0095 - val_securacy: 0.9264 - val_mese 0.           1. val_loss: 0.0095 - val_securacy: 0.9264 - val_mese 0.           1. val_loss: 0.0095 - val_securacy: 0.9264 - val_mese 0.           1. val_loss: 0.0095 - val_securacy: 0.9264 - val_mese 0.           1. val_loss: 0.0095 - val_securacy: 0.9264 - val_mese 0.           1. val_loss: 0.0095 - val_securacy: 0.9264 - val_mese 0.           1. val_loss: 0.0095 - val_securacy: 0.9264 - val_mese 0.           1. val_loss: 0.0095 - val_securacy: 0.9264 - val_mese 0.           1. val_loss: 0.0095 - val_securacy: 0.9264 - val_mese 0.           1. val_loss: 0.0095 - val_securacy: 0.9264 - val_mese 0.           1. val_loss: 0.0095 - val_securacy: 0.9264 - val_mese 0.           1. val_loss: 0.0095 - val_securacy: 0.9264 - val_mese 0.           1. val_loss: 0.0096 - val_securacy: 0.9264 - val_mese 0.           1. val_loss: 0.0096 - val_securacy: 0.9264 - val_mese 0.           1. val_loss: 0.00964 -
dense_5 (Dense) 	(None, 1) (None, 1) (a) LSTM M (b) LSTM M (c) LSTM (c)	101 Odel Sum testx, testy),validation, verboseil, shufflesfalee) racy: 0.9249 - mas: 0.005 racy: 0	splite.0.10,           1 - val_loss: 0.6074 - val_eccuracy: 0.5264 - val_mes: 0.           2 - val_loss: 0.6054 - val_eccuracy: 0.5264 - val_mes: 0.           3 - val_loss: 0.6056 - val_eccuracy: 0.5264 - val_mes: 0.           2 - val_loss: 0.6056 - val_eccuracy: 0.5264 - val_mes: 0.           2 - val_loss: 0.6056 - val_eccuracy: 0.5264 - val_mes: 0.           2 - val_loss: 0.6056 - val_eccuracy: 0.5264 - val_mes: 0.           2 - val_loss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_loss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_loss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_loss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_loss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_loss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_loss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_loss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_loss: 0.6055 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_loss: 0.6045 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_loss: 0.6045 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_loss: 0.6045 - val_eccuracy: 0.5264 - val_mes: 0.           1 - val_loss: 0.6045 - val_eccuracy: 0.5264 - val_mes: 0.
dense_5 (Dense) 	(None, 1) (None, 1) (a) LSTTM M (a) LSTM A (b) batch_sizes50, validation_dates (b) batch_sizes50, validation_dates (b) control (b) c	101 Odel Sum text, tertY, volidution, verbosen, shufflefleflar, racy: 0.9249 - mac: 0.008 racy: 0.9249 - mac: 0.008 racy: 0.9249 - mac: 0.006 racy: 0	splite0.10, splite0.10, splite0.10, a val_loss: 0.0074 - val_accuracy: 0.9264 - val_mes: 0. a val_loss: 0.0054 - val_accuracy: 0.9264 - val_mes: 0. a val_loss: 0.0054 - val_accuracy: 0.9264 - val_mes: 0. a val_loss: 0.0055 - val_accuracy: 0.9264 - val_mes: 0. a val_loss: 0.0055 - val_accuracy: 0.9264 - val_mes: 0. a val_loss: 0.0055 - val_accuracy: 0.9264 - val_mes: 0. a val_loss: 0.0055 - val_accuracy: 0.9264 - val_mes: 0. a val_loss: 0.0055 - val_accuracy: 0.9264 - val_mes: 0. 1 val_loss: 0.0055 - val_accuracy: 0.9264 - val_mes: 0. 1 val_loss: 0.0055 - val_accuracy: 0.9264 - val_mes: 0. 1 val_loss: 0.0055 - val_accuracy: 0.9264 - val_mes: 0. 1 val_loss: 0.0045 - val_accuracy: 0.9264 - val_mes: 0. 1 val_loss: 0.0045 - val_accuracy: 0.9264 - val_mes: 0. 1 val_loss: 0.0045 - val_accuracy: 0.9264 - val_mes: 0. 1 val_loss: 0.0045 - val_accuracy: 0.9264 - val_mes: 0. 1 val_loss: 0.0045 - val_accuracy: 0.9264 - val_mes: 0. 1 val_loss: 0.0045 - val_accuracy: 0.9264 - val_mes: 0. 1 val_loss: 0.0045 - val_accuracy: 0.9264 - val_mes: 0. 1 val_loss: 0.0045 - val_accuracy: 0.9264 - val_mes: 0. 1 val_loss: 0.0045 - val_accuracy: 0.9264 - val_mes: 0. 1 val_loss: 0.0045 - val_accuracy: 0.9264 - val_mes: 0. 1 val_loss: 0.0045 - val_accuracy: 0.9264 - val_mes: 0. 1 val_loss: 0.0045 - val_accuracy: 0.9264 - val_mes: 0. 1 val_loss: 0.0045 - val_accuracy: 0.9264 - val_mes: 0.
dense_5 (Dense) 	(None, 1) (None, 1) (a) LSTTM M hs=50, batch_size=50, validation_date(c) info(contor*val_less', patience:10)], (b) Jan/step - loss: 0.0083 - accu (c) 1015 Jan/step - loss: 0.0082 - accu (c) 1015 Jan/step - loss: 0.0082 - accu (c) 1015 Jan/step - loss: 0.0081 - accu	101 Odel Sum testX, testY),volidation, verboses1, shuffle=False) racy: 0.9249 - mae: 0.006 racy:	aplite.0.10,           3- val_loss: 0.6074 - val_accuracy: 0.5264 - val_mes: 0.           4. val_olss: 0.6054 - val_accuracy: 0.5264 - val_mes: 0.           5- val_olss: 0.6055 - val_accuracy: 0.5264 - val_mes: 0.           2 - val_olss: 0.6055 - val_accuracy: 0.5264 - val_mes: 0.           2 - val_olss: 0.6055 - val_accuracy: 0.5264 - val_mes: 0.           2 - val_olss: 0.6055 - val_accuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6055 - val_accuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6055 - val_accuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6055 - val_accuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6055 - val_accuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6055 - val_accuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6055 - val_accuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6055 - val_accuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6055 - val_accuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6051 - val_accuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6051 - val_accuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6045 - val_accuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6045 - val_accuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6045 - val_accuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6045 - val_accuracy: 0.5264 - val_mes: 0.           1 - val_olss: 0.6045 - val_accuracy: 0.5264 - val_mes: 0.
dense_5 (Dense) 	(None, 1) (None, 1) (a) LSTM M hs:59, batch_size59, validation_datas ping(sontor='val_loss', patience:10)]; -1 - 11s Jan/step - loss: 0.0003 - accu -1 - 10s Jan/step - loss: 0.0005 - accu	101 Odel Sum testX, testY),validation, verboseil, shuffletfalte) racy: 0.9249 - mae: 0.005 racy:	splite.0.10, splite.0.10, d - val_loss: 0.0074 - val_accuracy: 0.5264 - val_mes: 0. 6 - val_loss: 0.0054 - val_accuracy: 0.5264 - val_mes: 0. 7 - val_loss: 0.0056 - val_accuracy: 0.5264 - val_mes: 0. 7 - val_loss: 0.0055 - val_accuracy: 0.5264 - val_mes: 0. 7 - val_loss: 0.0055 - val_accuracy: 0.5264 - val_mes: 0. 7 - val_loss: 0.0055 - val_accuracy: 0.5264 - val_mes: 0. 8 - val_loss: 0.0055 - val_accuracy: 0.5264 - val_mes: 0. 8 - val_loss: 0.0055 - val_accuracy: 0.5264 - val_mes: 0. 9 - val_loss: 0.0055 - val_accuracy: 0.5264 - val_mes: 0. 1 - val_loss: 0.0055 - val_accuracy: 0.5264 - val_mes: 0. 1 - val_loss: 0.0055 - val_accuracy: 0.5264 - val_mes: 0. 1 - val_loss: 0.0055 - val_accuracy: 0.5264 - val_mes: 0. 1 - val_loss: 0.0055 - val_accuracy: 0.5264 - val_mes: 0. 1 - val_loss: 0.0051 - val_accuracy: 0.5264 - val_mes: 0. 1 - val_loss: 0.0054 - val_accuracy: 0.5264 - val_mes: 0. 1 - val_loss: 0.0054 - val_accuracy: 0.5264 - val_mes: 0. 1 - val_loss: 0.0054 - val_accuracy: 0.5264 - val_mes: 0. 1 - val_loss: 0.0054 - val_accuracy: 0.5264 - val_mes: 0. 1 - val_loss: 0.0055 - val_accuracy: 0.5264 - val_mes: 0. 1 - val_loss: 0.0055 - val_accuracy: 0.5264 - val_mes: 0. 1 - val_loss: 0.0055 - val_accuracy: 0.5264 - val_mes: 0. 1 - val_loss: 0.0055 - val_accuracy: 0.5264 - val_mes: 0. 1 - val_loss: 0.0055 - val_accuracy: 0.5264 - val_mes: 0. 1 - val_loss: 0.0055 - val_accuracy: 0.5264 - val_mes: 0. 1 - val_loss: 0.0055 - val_accuracy: 0.5264 - val_mes: 0. 1 - val_loss: 0.0055 - val_accuracy: 0.5264 - val_mes: 0.

#### (b) LSTM Model Fit

[128]:	<pre># moke predictions trainPredict = model.predict(trainX) testPredict = model.predict(testX)</pre>
[129]:	<pre>accr = model.evaluate(testX,testY) print('Test set\n Loss: {:0.3f}\n Accuracy: {:0.3f}'.format(accn[0],accn[1]))</pre>
	768/768 [ 0.9236 - mae: 0.0051 Test set Loss: 0.005 Accuracy: 0.924
[130]:	<pre># moke a prediction yhat = model.predict(testX) testX = testX.reshape((testX.shape[0], testX.shape[2]))</pre>

(c) LSTM Model Predict

Figure 18: Deep Learning Model Code Snippet Exp-1

#### 6.2 Experiment-2 LSTM Model Traffic and Non-traffic parameters

```
[166]: # create and fit the LSTM network
          # create and jet the import Bidirectional
model = Sequential()
#model.add(LSTM(100, activation='softmax', return_sequences=True, input_shape=(1, 40)))
          #model.add(LSTM(100, activation='softmax', return_sequences=irue, input_shape
#model.add(Bidirectional(LSTM(100, activation='relu'), input_shape=(1, 40)))
#model.add(LSTM(50, activation='relu'))
model.add(LSTM(100, activation='relu', input_shape=(1, 40))) # Vanilla- LSTM
model.add(Dropout(0.5))
          model.add(Dense(1))
          model.compile(loss='mae', optimizer='adam', metrics=['accuracy','mae'])
          model.summary()
          Model: "sequential_6"
          Layer (type)
                                                  Output Shape
                                                                                      Param #
                            ------
                                                 (None, 100)
          lstm 11 (LSTM)
                                                                                      56400
          dropout_6 (Dropout)
                                                 (None, 100)
                                                                                      0
          dense_6 (Dense)
                                                 (None, 1)
                                                                                      101
          Total params: 56,501
Trainable params: 56,501
          Non-trainable params: 0
```

(a) LSTM Model Summary

Epoch 1/50						
1889/1889 - 5s - lo	ss: 0.0046	- accuracy:	0.9329 - mae:	0.0046 - val_loss:	0.0039 - val_accuracy:	0.9355 - val_mae: 0.0039
Epoch 2/50						
1889/1889 - 5s - lo	ss: 0.0038	- accuracy:	0.9329 - mae:	0.0038 - val_loss:	0.0039 - val_accuracy:	0.9355 - val_mae: 0.0039
Epoch 3/50						
1889/1889 - 5s - lo	ss: 0.0037	- accuracy:	0.9329 - mae:	0.0037 - val_loss:	0.0039 - val_accuracy:	0.9355 - val_mae: 0.0039
Epoch 4/50						
1889/1889 - 5s - lo	ss: 0.0036	- accuracy:	0.9329 - mae:	0.0036 - val_loss:	0.0039 - val_accuracy:	0.9355 - val_mae: 0.0039
Epoch 5/50						
1889/1889 - 5s - lo	ss: 0.0036	- accuracy:	0.9329 - mae:	0.0036 - val_loss:	0.0036 - val_accuracy:	0.9355 - val_mae: 0.0036
Epoch 6/50						
1889/1889 - 6s - lo	ss: 0.0036	- accuracy:	0.9329 - mae:	0.0036 - val_loss:	0.0036 - val_accuracy:	0.9355 - val_mae: 0.0036
Epoch 7/50						
1889/1889 - 5s - lo	ss: 0.0036	- accuracy:	0.9329 - mae:	0.0036 - val_loss:	0.0036 - val_accuracy:	0.9355 - val_mae: 0.0036
Epoch 8/50						
1889/1889 - 5s - Ic	ss: 0.0036	- accuracy:	0.9329 - mae:	0.0036 - val_loss:	0.0037 - val_accuracy:	0.9355 - val_mae: 0.0037
Epoch 9/50						
1889/1889 - 5s - 1c	ss: 0.0036	- accuracy:	0.9329 - mae:	0.0036 - val_loss:	0.0036 - val_accuracy:	0.9355 - val_mae: 0.0036
Epoch 10/50						
1889/1889 - 6s - 1d	ss: 0.0035	- accuracy:	0.9329 - mae:	0.0035 - val_loss:	0.0035 - val_accuracy:	0.9355 - val_mae: 0.0035
Epoch 11/50	0 0000				0.0000	0.0355 3 0.0035
1889/1889 - 55 - 10	ISS: 0.0036	- accuracy:	0.9329 - mae:	0.0036 - Val_1055:	0.0036 - Val_accuracy:	0.9355 - Val_mae: 0.0036
Epoch 12/50			0.0320	0.0005	0.0036	0.0355
1009/1009 - 55 - 10 Easth 13/50	55: 0.0055	- accuracy:	0.9529 - mae:	0.0055 - Val_1055:	0.0056 - Val_accuracy:	0.9555 - Val_mae: 0.0056
1000/1000 Ec 1c			0.0220	0.0026	0.0040	0.0355 vol moor 0.0040
Epoch 14/59	55. 0.0000	- accuracy.	0.9329 - mae.	0.0050 - Val_1055.	0.0040 - Val_accuracy.	0.9555 - Val_mae. 0.0040
1880/1880 - 5r - 1r	a aass		0 9329 - mao:	0 0035 - val locc	0 0036 - val accupacy:	0 9355 - val mao: 0 0036
Enoch 15/50		accuracy.	0.5525 mae.	0.0055 081_1055.	var_accuracy.	0.5555 Var_mae. 0.0050
1889/1889 - 54 - 16	A AA35	- accuracy:	0 9329 - mao.	0 0035 - val loss:	0 0037 - val accuracy:	0 9355 - val mae: 0 0037
Enoch 16/50		222010091	moer			
1889/1889 - 55 - 10	ss: 0.0035	- accuracy:	0.9329 - mae:	0.0035 - val loss:	0.0037 - val accuracy:	0.9355 - val mae: 0.0037
Enoch 17/50		222010091	moer			

#### (b) LSTM Model Fit

[168]:	<pre># make predictions trainPredict = model.predict(trainX) testPredict = model.predict(testX)</pre>								
[78]:	<pre>accr = model.evaluate(testX,testY) print('Test set\n Loss: {:0.3f}\n Accuracy: {:0.3f}'.format(accr[0],accr[1]))</pre>								
	118/118 [===================================								
[363]:	<pre># make a prediction yhat = model.predict(testX)</pre>								
[364]:	<pre>testX = testX.reshape((testX.shape[0], testX.shape[2]))</pre>								
[365]:	<pre>from numpy import concatenate # invert scaling for forecast #inv_yhat = concatenate((yhat[:,0,:], testX[:, 0:]), axis=1) inv_yhat = concatenate((yhat, testX[:, 0:]), axis=1) inv_yhat = scaler.inverse_transform(inv_yhat) inv_yhat = inv_yhat[:,0]</pre>								

(c) LSTM Model Predict

Figure 19: Deep Learning Model Code Snippet Exp-2

#### 6.3 Experiment-3 CNN Model Traffic-only parameters

CNN model with 1 convolutional layer, 1 pooling layer and 3 hidden layers are applied to the traffic speed limit data.

[126]:	<pre>[126]: model = Sequential() intput_shape=(x_train.shape[], 1) model.add(ConvUD(128, kernel_size=3,padding = 'same',activation='relu', input_shape=input_shape model.add(Km2NoolingID(pool_size=(2))) model.add(Cherse(64, activation='tanh')) model.add(Cherse(52, activation='telu')) model.add(Cherse(15, activation='relu')) model.adm2 </pre>										
	Model: "sequential_3"										
	Layer (type)	Output Shape	Param #								
	conv1d_3 (Conv1D)	(None, 13, 128)	512								
	batch_normalization_3 (Batch	h (None, 13, 128)	512								
	<pre>max_pooling1d_3 (MaxPooling:</pre>	0									
	flatten_3 (Flatten)	(None, 768)	0								
	dense_12 (Dense)	(None, 64)	49216								
	dropout_9 (Dropout)	(None, 64)	0								
	dense_13 (Dense)	(None, 32)	2080								
	dropout_10 (Dropout)	(None, 32)	0								
	(a) CNN Model Summary										
[127]:	<pre>from keras.optimizers import SGD, RMSprop #opt = SGO((-m0.01, momentum=0.9) opt = RMSprop(ln=0.001) model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy']) bictarum = model fifty trains a tenin = merchangenetic bick in all divide district tents to tent.</pre>										
	history = model.fit(x_train, y_train, epochs:epochs, batch_size:batch_size, validation_data(x_test, y_test),validation_split=0.10, callbacks=[EarlyStopping(monitors'val_loss', patience=10]], verbose=2, shuffle=False) Epoch 1/50 736/736 - 3s - loss: 0.1490 - accuracy: 0.9418 - val_loss: 0.0435 - val_accuracy: 0.9406 Epoch 2/50 736/736 - 3s - loss: 0.0527 - accuracy: 0.9412 - val_loss: 0.0437 - val_accuracy: 0.9606 Epoch 3/50 736/736 - 3s - loss: 0.0456 - accuracy: 0.9847 - val_loss: 0.0437 - val_accuracy: 0.9829 Epoch 4/50 736/736 - 3s - loss: 0.0456 - accuracy: 0.9847 - val_loss: 0.0327 - val_accuracy: 0.9858 Epoch 5/50 736/736 - 3s - loss: 0.0456 - accuracy: 0.9847 - val_loss: 0.0327 - val_accuracy: 0.9860 Epoch 6/50 736/736 - 3s - loss: 0.0456 - accuracy: 0.9867 - val_loss: 0.0327 - val_accuracy: 0.9860 Epoch 6/50 736/736 - 3s - loss: 0.0356 - accuracy: 0.9867 - val_loss: 0.0327 - val_accuracy: 0.9868 Epoch 6/50 736/736 - 3s - loss: 0.0376 - accuracy: 0.9869 - val_loss: 0.0359 - val_accuracy: 0.9868 Epoch 6/50 736/736 - 3s - loss: 0.0376 - accuracy: 0.9869 - val_loss: 0.0359 - val_accuracy: 0.9868 Epoch 10/50 736/736 - 3s - loss: 0.0376 - accuracy: 0.9867 - val_loss: 0.0359 - val_accuracy: 0.9868 Epoch 10/50 736/736 - 3s - loss: 0.0371 - accuracy: 0.9867 - val_loss: 0.0359 - val_accuracy: 0.9868 Epoch 11/50 736/736 - 3s - loss: 0.0374 - accuracy: 0.9871 - val_loss: 0.0359 - val_accuracy: 0.9868 Epoch 11/50 736/736 - 3s - loss: 0.0372 - accuracy: 0.9866 - val_loss: 0.0359 - val_accuracy: 0.9868 Epoch 13/50 736/736 - 3s - loss: 0.0372 - accuracy: 0.9866 - val_loss: 0.0359 - val_accuracy: 0.9868 Epoch 13/50 736/736 - 3s - loss: 0.0372 - accuracy: 0.9866 - val_loss: 0.0359 - val_accuracy: 0.9868 Fpoch 13/50 736/736 - 3s - loss: 0.0372 - accuracy: 0.9866 - val_loss: 0.0359 - val_accuracy: 0.9868 Fpoch 13/50 736/736 - 3s - loss: 0.0372 - accuracy: 0.9866 - val_loss: 0.0359 - val_accuracy: 0.9868 Fpoch 13/50 736/736 - 3s - loss: 0.0372 - accuracy: 0.9866 - val_loss: 0.0359 - val_a										
[129]:	<pre>y_pred = model.predict( score = model.evaluate(</pre>	x_test, batch_size=10, verbose=1) x_test, y_test,verbose:	=1)								
	print(score)										
	454/454 [] - 9c 769us/stop										

(c) CNN Model Predict



## 6.4 Experiment-4 CNN Model Traffic and Non-traffic parameters

41]:	model = Sequential()												
	intput_shape=(x_train.shape[	1], 1)											
	<pre>model.add(Conv1D(128, kernel_size=3,padding = 'same',activation='relu', input_shape=input_shape))</pre>												
	<pre>model.add(BatchNormalization())</pre>												
	<pre>model.add(MaxPooling1D(pool_size=(2)))</pre>												
	<pre>model.add(Flatten())</pre>												
	model.add(Dense(64, activati	on='tan	h'))										
	<pre>model.add(Dropout(0.5))</pre>												
	<pre>model.add(Dense(32, activation='relu'))</pre>												
	<pre>model.add(Dropout(0.5))</pre>												
	<pre>model.add(Dense(16, activation='relu'))</pre>												
	<pre>model.add(Dropout(0.5))</pre>												
	<pre>model.add(Dense(num_classes, activation='softmax'))</pre>												
	<pre>model.summary()</pre>												
	Medel, "sequential"												
	Model: Sequencial												
	Layer (type)	Output	Shape	Param #									
	conv1d (Conv1D)	(None,	40, 128)	512									
		· · ·											
	batch_normalization (BatchNo	(None,	40, 128)	512									
	may peoling1d (MayPeoling1D)	(None	20 128)	0									
	max_pooringid (maxpooringid)	(None,	20, 120)	0									
	flatten (Flatten)	(None	2560)	0									
	ridecen (ridecen)	(110110)	2500)	0									
	dense (Dense)	(None,	64)	163904									
			·										
	dropout (Dropout)	(None,	64)	0									
	dense_1 (Dense)	(None,	32)	2080									
	dropout_1 (Dropout)	(None,	32)	0									
		(1)	46	500									
	dense_2 (Dense)	(None,	10)	528									

#### (a) CNN Model Summary

	<pre>#opt = SOD(Ir=0.01, momentum=0.9) model.compile(loss='ategorical_crossentropy', optimizer='adam', metrics=['accuracy']) #model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])</pre>										
[43]:	<pre>history = model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size, validation_data=(x_test, y_test),validation_split=0.10,</pre>										
	Epoch 1/50										
	2042/2042 - 15s - loss: 0.0634 - accuracy: 0.9794 - val_loss: 0.0162 - val_accuracy: 0.9958										
	Epoch 2/50										
	2042/2042 - 14s - loss: 0.0207 - accuracy: 0.9952 - val_loss: 0.0161 - val_accuracy: 0.9958										
	Epoch 3/50										
	2042/2042 - 14s - loss: 0.0196 - accuracy: 0.9948 - val_loss: 0.0162 - val_accuracy: 0.9958										
	Epoch 4/50										
	2042/2042 - 14s - loss: 0.0180 - accuracy: 0.9952 - val_loss: 0.0162 - val_accuracy: 0.9958										
	Epoch 5/50										
	2042/2042 - 14s - loss: 0.0170 - accuracy: 0.9954 - val loss: 0.0161 - val accuracy: 0.9958										

2042/2042	749		1033.	0.01/0		accuracy.	0.2224		ver_1033.	0.0101		ver_acconacy.	0.5550
Epoch 6/50													
2042/2042 -	14s	-	loss:	0.0171	-	accuracy:	0.9953	-	val_loss:	0.0162	-	val_accuracy:	0.9958
Epoch 7/50													
2042/2042 -	14s	-	loss:	0.0173	-	accuracy:	0.9953	-	val_loss:	0.0161	-	val_accuracy:	0.9958
Epoch 8/50													
2042/2042 -	14s	-	loss:	0.0167	-	accuracy:	0.9956	-	val_loss:	0.0162	-	val_accuracy:	0.9958
Epoch 9/50													
2042/2042 -	14s	-	loss:	0.0180	-	accuracy:	0.9952	-	val_loss:	0.0161	-	val_accuracy:	0.9958
Epoch 10/50													
2042/2042 -	14s	-	loss:	0.0170	-	accuracy:	0.9954	-	val_loss:	0.0161	-	val_accuracy:	0.9958
Epoch 11/50													
2042/2042 -	15s	-	loss:	0.0168	-	accuracy:	0.9954	-	val_loss:	0.0162	-	val_accuracy:	0.9958
Epoch 12/50													
2042/2042 -	16s	-	loss:	0.0168	-	accuracy:	0.9954	-	val_loss:	0.0162	-	val_accuracy:	0.9958
Epoch 13/50													
2042/2042 -	17s	-	loss:	0.0166	-	accuracy:	0.9955	-	val_loss:	0.0162	-	val_accuracy:	0.9958
Epoch 14/50													
2042/2042 -	14s	-	loss:	0.0163	-	accuracy:	0.9956	-	val_loss:	0.0162	-	val_accuracy:	0.9958
Enoch 15/50													

[42]: from keras.optimizers import SGD

#### (b) CNN Model Fit

(c) CNN Model Predict

Figure 21: Deep Learning Model Code Snippet Exp-4

## 7 Evaluation Output

Finally, the models are evaluated using training and testing accuracy, RMSE and MAE values from **sklearn.metrics** and confusion matrix from **scikitplot**. Also, the model's training and validation loss against the number of epochs is visualized to check whether the model is a overfit or underfit or perfect fit.

#### 7.1 Experiment-1 Evaluation Code Snippet



(a) Evaluation Code

(b) Loss Vs Number of Epoch Code



#### 7.2 Experiment-2 Evaluation Code Snippet



Figure 23: Evaluation Code Snippet Exp-2

#### 7.3 Experiment-3 Evaluation Code Snippet



(a) Loss Vs Number of Epoch Code

(b) Confusion Matrix Code



#### 7.4 Experiment-4 Evaluation Code Snippet



(a) Loss Vs Number of Epoch Code

(b) Confusion Matrix Code

Figure 25: Evaluation Code Snippet Exp-4

## References

- Data Engineering with Google Cloud Professional Certificate (n.d.). https://www.coursera.org/professional-certificates/gcp-data-engineering#courses. Accessed: 2020-07-15.
- Google Cloud Platform (GCP) documentation (n.d.). https://cloud.google.com/docs. Accessed: 2020-08-10.
- Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization (n.d.). https://www.coursera.org/learn/deep-neural-network/home/ welcome. Accessed: 2020-07-29.