Deep Learning on Traffic Prediction Methods Analysis and Future Directions

A PROJECT REPORT

Submitted by

NURUKURTHI VEERA VENKATA GANESH
[Reg No: RA1911027010109]
SRI SURYA HEMANTH BALUSU
[Reg No: RA1911027010095]

Under the Guidance of

Dr. T VEERAMAKALI

Associate Professor, Department of Data Science and Business Systems *In partial fulfilment of the requirements for the degree of*

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING
with a specialization in BIG DATA ANALYTICS

DEPARTMENT OF DATA SCIENCE AND BUSINESS SYSTEMS
COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR-603203
MAY 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this B.Tech project report titled "**Deep Learning on Traffic Prediction Method Analysis and Future Directions**" is the bonafide work of Mr. **Nurukurthi Veera Venkata Ganesh [Reg No: RA1911027010109]** and Mr. **Balusu Sri Surya Hemanth [Reg No: RA1911027010095]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

Signature

Dr. T. Veeramakali

SUPERVISOR

Associate Professor

Data Science and Business Systems

Signature

Dr. M. Lakshmi

HEAD OF THE DEPARTMENT

Data Science and Business Systems

SIGNATURE OF INTERNAL EXAMINER

SIGNATURE OF EXTERNAL EXAMINER

Department of Data Science and Business Systems

SRM Institute of Science and Technology

Own Work Declaration Form

Degree/ Course : B.Tech in Computer Science and Engineering with a specialization in Big Data Analytics

Student Names : **Nurukurthi Veera Venkata Ganesh, Balusu Sri Surya Hemanth**

Registration Number: RA1911027010109, **RA1911027010095**

Title of Work : **Deep Learning on Traffic Prediction Method Analysis and Future Directions**

We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines.

We confirm that all the work contained in this assessment is our own except where indicated, and that this research has met the following conditions:

- Clearly references / listed all sources as appropriate.
- Referenced and put in inverted commas all quoted text (from books, web, etc.)
- Given the sources of all pictures, data etc. that are not my own.

- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g., fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website.

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

RA1911027010095
BALUSU SRI SURYA HEMANTH

RA1911027010109
NURUKURTHI VEERA VENKATA GANESH

ACKNOWLEDGEMENT

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to Dean-CET, SRM Institute of Science and Technology, **Dr. T.V.Gopal**, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We are incredibly grateful to our Head of the Department, **Dr. M. Lakshmi,** Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our Project Coordinator, **Dr. P. Rajasekar**, Assistant Professor, Panel Head, **Dr. A. Murugan**, Associate Professor and members, **Dr. R. Rajkumar,** Assistant Professor, **Dr. T. Karthick,** Assistant Professor and **Dr. S. Jeeva,** Assistant Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for their inputs during the project reviews and support.

We register our immeasurable thanks to our Faculty Advisor, **Dr. K. Priyadarsini**, Assistant Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide, **Dr. A. Murugan**, Associate Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under his mentorship. He provided us with the freedom and support to explore the research topics of our interest. His passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank the Data Science and Business Systems Department staff and students, SRM Institute of Science and Technology, for their help during our project. Finally, we would like to thank parents, family members, and friends for their unconditional love, constant support, and encouragement.

Nurukurthi Veera Venkata Ganesh

Balusu Sri Surya Hemanth

TABLE OF CONTENTS

ABSTRACT

Traffic prediction plays an essential role in intelligent transportation system. Accurate traffic prediction can assist route planing, guide vehicle dispatching, and mitigate traffic congestion. This problem is challenging due to the complicated and dynamic spatio-temporal dependencies between different regions in the road network. Recently, a significant amount of research efforts have been devoted to this area, especially deep learning method, greatly advancing traffic prediction abilities. The purpose of this paper is to provide a comprehensive survey on deep learning-based approaches in traffic prediction from multiple perspectives. Specifically, we first summarize the existing traffic prediction methods, and give a taxonomy. Second, we list the state-of-the-art approaches in different traffic prediction applications. Third, we comprehensively collect and organize widely used public datasets in the existing literature to facilitate other researchers. Furthermore, we give an evaluation and analysis by conducting extensive experiments to compare the performance of different methods on a real-world public dataset. Finally, we discuss open challenges in this field.

LIST OF FIGURES

ABBREVIATIONS

ABBREVIATIONS	MEANING
STGCN	Spatial Temporal Graph Convolutional Network
STGCN	Spatial Temporal Graph Convolutional Network
STSGCN	Magnetic Resonance Imaging
ASTGCN	Attention-based Spatial Temporal Graph Convolutional Network
Graph WaveNet	Graph Wavelet Neural Network
CNN	Convolutional Neural Network
CNN+RNN	Convolutional Neural Network + Recurrent Neural Network
DCRNN	Diffusion Convolutional Recurrent Neural Network
SVM	Support Vector Machine
GMAN	Gated Multi-scale Aggregation Network

CHAPTER 1

INTRODUCTION

The modern city is gradually developing into a smart city. The acceleration of urbanization and the rapid growth of urban population bring great pressure to urban traffic management. Intelligent Transportation System (ITS) is an indispensable part of smart city, and traffic prediction is an important component of ITS. Accurate traffic prediction is essential to many real-world applications. For example, traffic flow prediction can help city alleviate congestion; car-hailing demand prediction can prompt car-sharing companies to preallocate cars to high demand regions. The growing available traffic related datasets provide us potential new perspectives to explore this problem.

1.1 SOFTWARE REQUIREMENTS

Software requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application. These requirements or prerequisites are generally not included in the software installation package and need to be installed separately before the software is installed.

Platform – In computing, a platform describes some sort of framework, either in hardware or software, which allows software to run. Typical platforms include a computer's architecture, operating system, or programming languages and their runtime libraries.

Operating system is one of the first requirements mentioned when defining system requirements (software). Software may not be compatible with different versions of same line of operating systems, although some measure of backward compatibility is often maintained. For example, most software designed for Microsoft Windows XP does not run on Microsoft Windows 98, although the converse is not always true. Similarly, software designed using newer features of Linux Kernel v2.6 generally does not run or compile properly (or at all) on Linux distributions using Kernel v2.2 or v2.4.

APIs and drivers – Software making extensive use of special hardware devices, like high-end display adapters, needs special API or newer device drivers. A good example is DirectX, which is a collection of APIs for handling tasks related to multimedia, especially game programming, on Microsoft platforms.

Web browser – Most web applications and software depending heavily on Internet technologies make use of the default browser installed on system. Microsoft Internet Explorer is a frequent choice of software running on Microsoft Windows, which makes use of ActiveX controls, despite their vulnerabilities.

1)Visual Studio Community Version

2)Nodejs (Version 12.3.1)

3)Python IDEL (Python 3.7)

1.2 HARDWARE REQUIREMENTS

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware, A hardware requirements list is often accompanied by a hardware compatibility list (HCL), especially in case of operating systems. An HCL lists tested, compatible, and sometimes incompatible hardware devices for a particular operating system or application. The following sub-sections discuss the various aspects of hardware requirements.

Architecture – All computer operating systems are designed for a particular computer architecture. Most software applications are limited to particular operating systems running on particular architectures. Although architecture-independent operating systems and applications exist, most need to be recompiled to run on a new architecture. See also a list of common operating systems and their supporting architectures.

Processing power – The power of the central processing unit (CPU) is a fundamental system requirement for any software. Most software running on x86 architecture define processing power as the model and the clock speed of the CPU. Many other features of a CPU that influence its speed and power, like bus speed, cache, and MIPS are often ignored. This definition of power is often erroneous, as AMD Athlon and Intel Pentium CPUs at similar clock speed often have different throughput speeds. Intel Pentium CPUs have enjoyed a considerable degree of popularity, and are often mentioned in this category.

Memory – All software, when run, resides in the random access memory (RAM) of a computer. Memory requirements are defined after considering demands of the application, operating system, supporting software and files, and other running processes. Optimal performance of other unrelated software running on a multi-tasking computer system is also considered when defining this requirement.

Secondary storage – Hard-disk requirements vary, depending on the size of software installation, temporary files created and maintained while installing or running the software, and possible use of swap space (if RAM is insufficient).

Display adapter – Software requiring a better than average computer graphics display, like graphics editors and high-end games, often define high-end display adapters in the system requirements.

Peripherals – Some software applications need to make extensive and/or special use of some peripherals, demanding the higher performance or functionality of such peripherals. Such peripherals include CD-ROM drives, keyboards, pointing devices, network devices, etc.

1)Operating System : Windows Only

2)Processor : i5 and above

3)Ram : 4gb and above

4)Hard Disk : 50 GB

CHAPTER 2

FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- ECONOMICAL FEASIBILITY
- TECHNICAL FEASIBILITY
- SOCIAL FEASIBILITY

2.1 ECONOMICAL FEASIBILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

2.2 TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

2.3 SOCIAL FEASIBILITY

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

CHAPTER 3

LITERATURE SURVEY

3.1 DNN-based prediction model for spatio-temporal data:

The paper proposes a deep-learning-based prediction model, called DeepST, for spatio-temporal (ST) data that has unique spatial and temporal properties. The model comprises two components: spatio-temporal and global. The spatio-temporal component uses convolutional neural networks to model spatial dependencies and temporal trends. The global component captures global factors such as day of the week. The model is used to build a real-time crowd flow forecasting system called UrbanFlow1, and experiment results on diverse ST datasets show its ability to capture ST data's spatio-temporal properties and outperform four baseline methods.[1]

3.2 Short-term traffic forecasting: Where we are and where we're going

The paper discusses the history and current state of short-term traffic forecasting in Intelligent Transportation Systems (ITS). Existing literature has focused on using single-point data from motorways and univariate mathematical models to predict traffic volumes or travel times. Recent technological advancements and computational power present opportunities to expand research into under-researched areas. The paper reviews existing challenges and suggests ten directions for future work. [2]

3.3 A brief overview of machine learning methods for short-term traffic forecasting and future directions

The paper provides an overview of machine learning approaches for short-term traffic forecasting in intelligent transportation systems. It highlights the recent advancements in the field due to data availability and machine learning techniques. The paper introduces the challenges of traffic forecasting and various approaches for modeling temporal and spatial dependencies. It concludes by discussing important future research directions.[3]

3.4 Survey on traffic prediction in smart cities

The paper provides a comprehensive survey of traffic prediction for intelligent transportation systems, focusing on the spatio-temporal data layer to the intelligent transportation application layer. It splits the research scope into four parts, including spatio-temporal data, preprocessing, traffic prediction, and traffic application, and reviews existing work on each part. It summarizes traffic data into five types, data preprocessing techniques, and three kinds of traffic prediction problems. The paper also lists five typical traffic applications and provides emerging research challenges and opportunities. The survey aims to help practitioners understand existing traffic prediction problems and methods to solve intelligent transportation applications.[4]

3.5 Machine learning-based traffic prediction models for intelligent transportation systems

The paper discusses the importance of traffic flow prediction in the field of Intelligent Transportation Systems (ITS), which can lead to safer, more efficient, and enjoyable transportation environments. Machine learning (ML) methods have been widely used in traffic flow prediction due to their ability to fit non-linear features in traffic data. The paper provides a thorough review of different ML models, categorized based on the ML theory they use. The advantages and disadvantages of each model are analyzed, and comparisons are made among different categories to help identify which ML methods are good for specific prediction tasks. The paper also reviews the useful add-ons used in traffic prediction and discusses open challenges in the field. [5]

CHAPTER 4

SYSTEM ANALYSIS

4.1 EXISTING SYSTEM:

Alexander et al. presented a survey of deep neural network for traffic prediction. It discussed three common deep neural architectures, including convolutional neural network, recurrent neural network, and feedforward neural network. However, some recent advancements, e.g., graph-based deep learning, were not covered. An overview of graph-based deep learning architecture, with applications in the general traffic domain. Authors provided a survey focusing specifically on the use of deep learning models for analyzing traffic data. However, it only investigates the traffic flow prediction. In general, different traffic prediction tasks have common characteristics, and it is beneficial to consider them jointly. Therefore, there is still a lack of broad and systematic survey on exploring traffic prediction in general.

4.1.1 DISADVANTAGES OF EXISTING SYSTEM:

1. There is still a lack of broad and systematic survey on exploring traffic prediction in general.
2. Accurate traffic prediction can assist route planing, guide vehicle dispatching, and mitigate traffic congestion. This problem is challenging due to the complicated and dynamic spatio-temporal dependencies between different regions in the road network.

4.2 Proposed System:

The purpose of this paper is to provide a comprehensive survey on deep learning-based approaches in traffic prediction from multiple perspectives. Specifically, we first summarize the existing traffic prediction methods, and give a taxonomy. Second, we list the state-of-the-art approaches in different traffic prediction applications. Third, we comprehensively collect and organize widely used public datasets in the existing literature to facilitate other researchers. Furthermore, we give an evaluation and analysis by conducting extensive experiments to compare the performance of different methods on a real-world public dataset. Finally, we discuss open challenges in this field.

4.2.1 Advantages of proposed system:

1. This paper is suitable for participators to quickly understand the traffic prediction, so as to find branches they are interested in.

2. It also provides a good reference and inquiry for researchers in this field, which can facilitate the relevant research.

4.3 FUNCTIONAL REQUIREMENTS

1.Data Collection

2.Data Preprocessing

3.Training And Testing

4.Modiling

5.Predicting

4.4 NON FUNCTIONAL REQUIREMENTS

NON-FUNCTIONAL REQUIREMENT (NFR) specifies the quality attribute of a software system. They judge the software system based on Responsiveness, Usability, Security, Portability and other non-functional standards that are critical to the success of the software system. Example of nonfunctional requirement, *"how fast does the website load?"* Failing to meet non-functional requirements can result in systems that fail to satisfy user needs. Non-functional Requirements allows you to impose constraints or restrictions on the design of the system across the various agile backlogs. Example, the site should load in 3 seconds when the number of simultaneous users are > 10000. Description of non-functional requirements is just as critical as a functional requirement.

- Usability requirement
- Serviceability requirement
- Manageability requirement
- Recoverability requirement
- Security requirement
- Data Integrity requirement
- Capacity requirement

- Availability requirement
- Scalability requirement
- Interoperability requirement
- Reliability requirement
- Maintainability requirement
- Regulatory requirement
- Environmental requirement

CHAPTER 5

SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE:

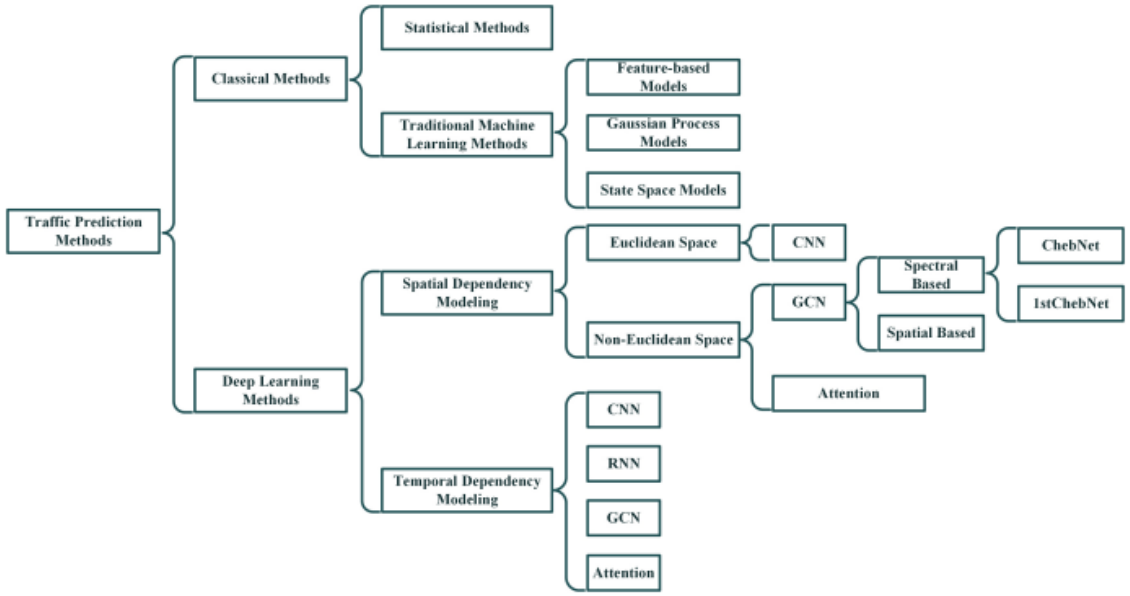


Fig.5.1.1 System architecture

DATA FLOW DIAGRAM:

1. The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.
2. The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.
3. DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.

4. DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.

5. Data flow diagrams are commonly used in systems analysis and design to model and understand the flow of data within a system. They help visualize how data moves through processes, identifies inputs and outputs, and highlights interactions between different components.

6. By creating a DFD, analysts can gain insights into the system's data requirements, dependencies, and potential bottlenecks. It can also aid in identifying areas for improvement, optimizing data flow, and enhancing system performance

7. DFDs can be created at various levels of detail, ranging from high-level overviews to more granular representations of individual processes. They provide a clear and concise representation of data movement, making them useful for communication and documentation purposes during system development and maintenance.

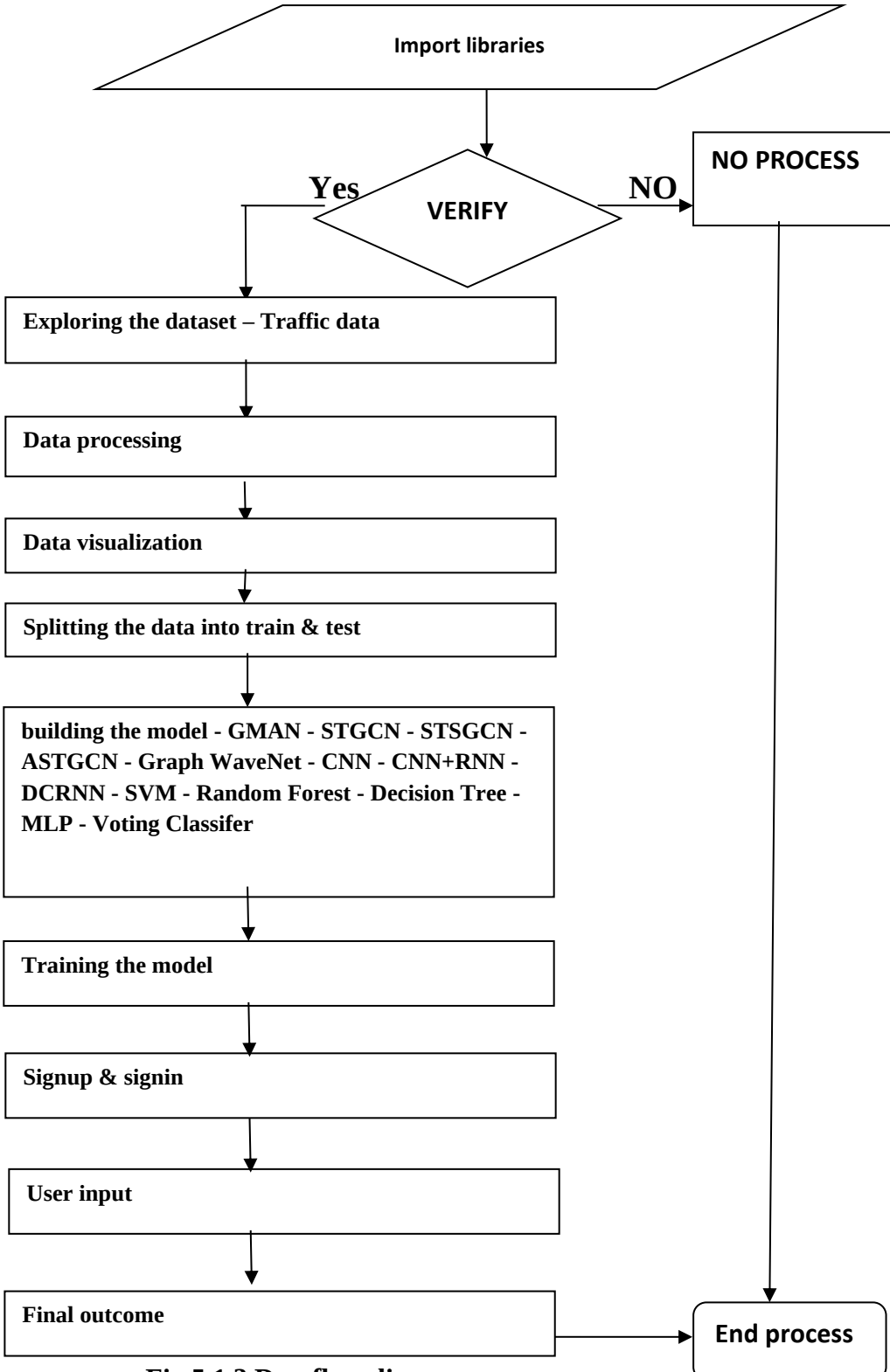


Fig.5.1.3 Dataflow diagram

5.2 UML DIAGRAMS

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

GOALS:

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

Use case diagram:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

In a use case diagram, the primary elements are

1. Actors: Actors represent the different types of users or external entities that interact with the system. They are depicted as stick figures or named blocks on the diagram. Actors can be individuals, other systems, or even hardware devices.
2. Use Cases: Use cases represent specific tasks, actions, or functionalities that the system provides to its users. They describe the system's behavior from the user's perspective. Use cases are represented as ovals on the diagram and are connected to the actors that initiate or participate in those use cases.
3. Relationships: Relationships between actors and use cases are depicted by lines connecting them. The main types of relationships in use case diagrams include:

    - Association: It represents a general interaction between an actor and a use case.
    - Include: It signifies that one use case includes the functionality of another use case.
    - Extend: It indicates that one use case can be extended by another use case.
    - Generalization: It represents an inheritance relationship between actors or use cases, indicating that one is a specialized form of another.

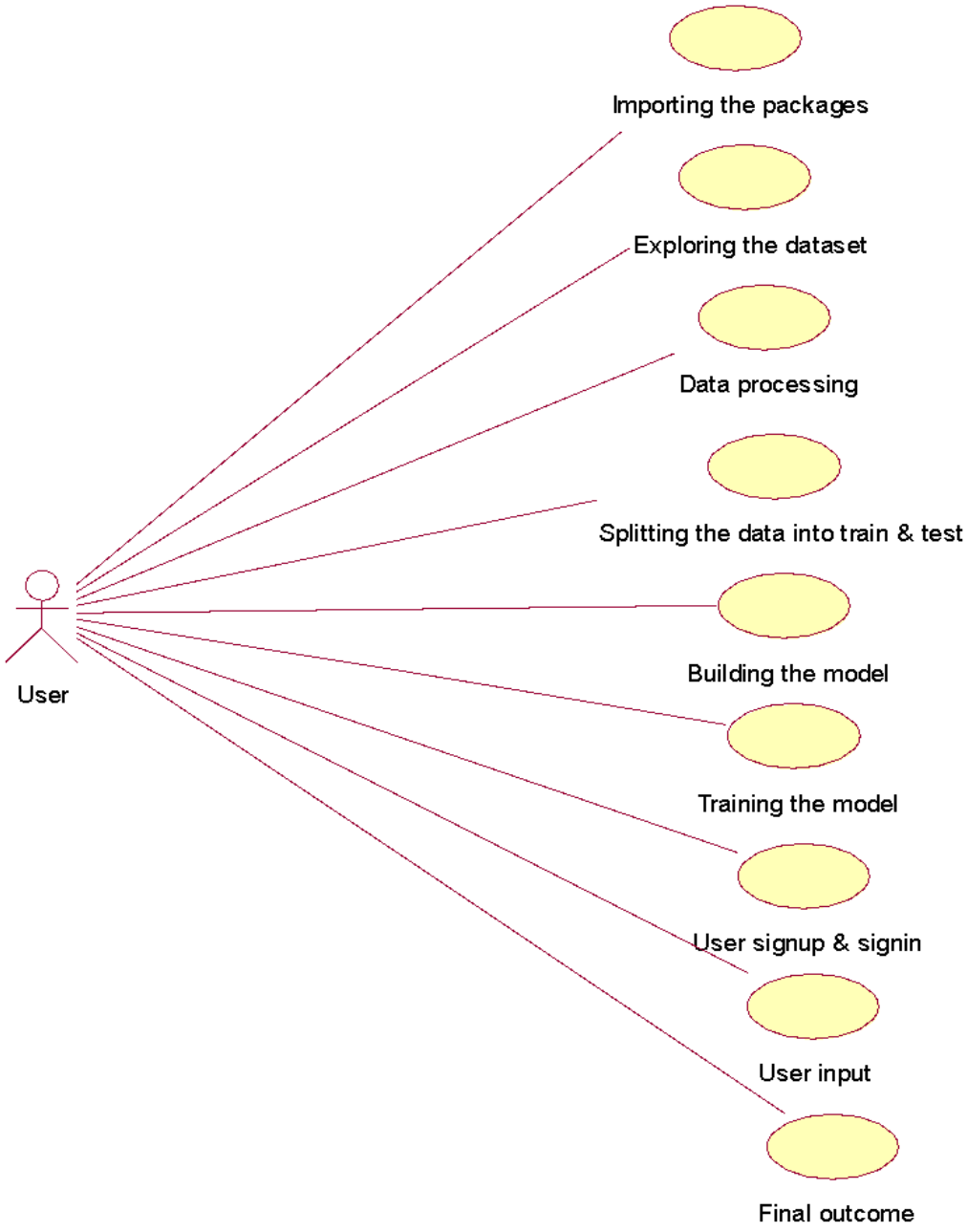


Fig.5.2.1 Usecase diagram

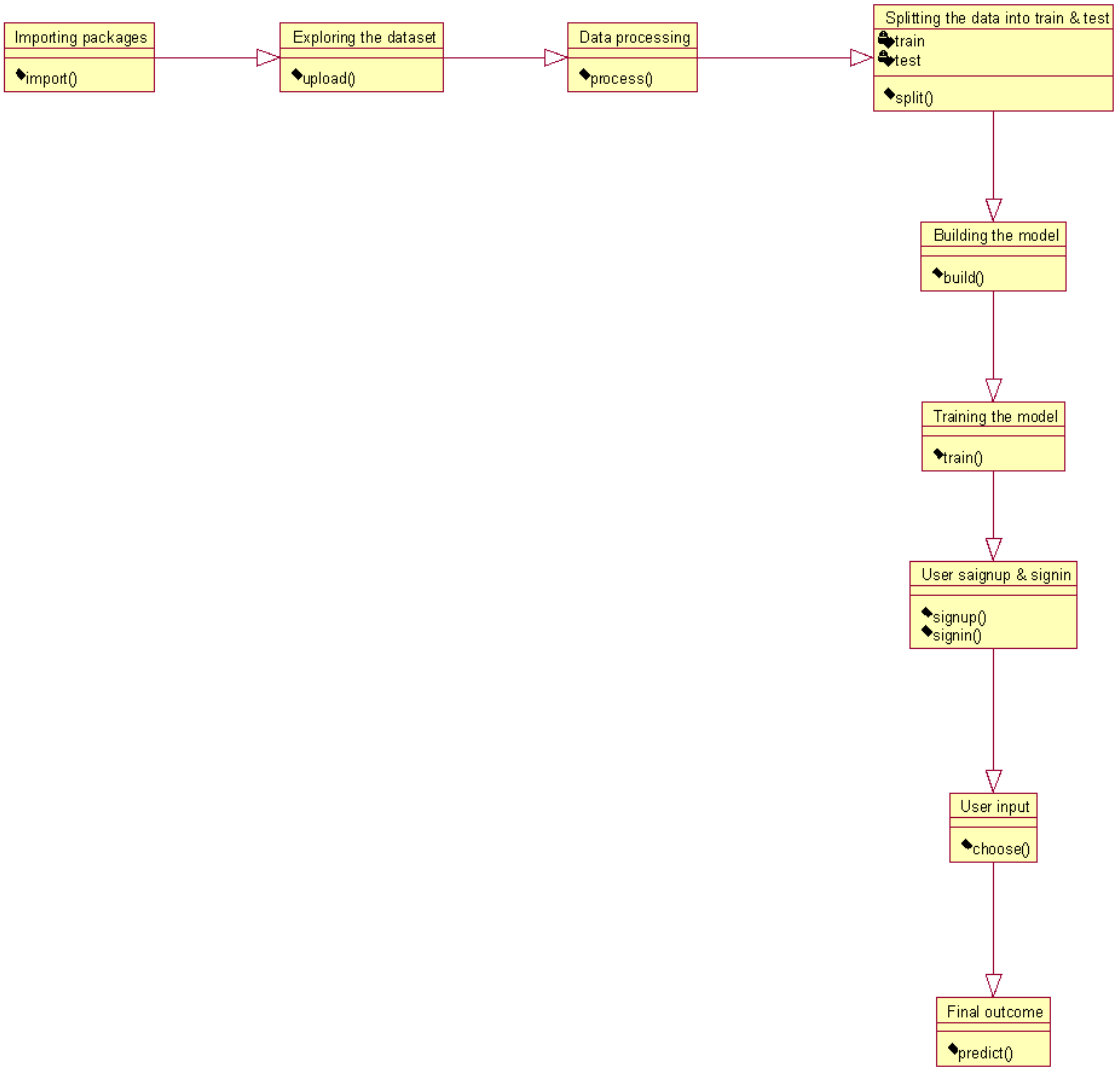
Class diagram:

The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an "is-a" or "has-a" relationship. Each class in the class diagram may be capable of providing certain functionalities. These functionalities provided by the class are termed "methods" of the class. Apart from this, each class may have certain "attributes" that uniquely identify the class.

In a class diagram, the primary elements are:

- Class: A class represents a blueprint or template for creating objects. It encapsulates attributes (data) and methods (behavior) that define the characteristics and operations of objects belonging to that class. Classes are depicted as rectangles, with the class name at the top, followed by attributes and methods listed beneath.
- Attributes: Attributes are the properties or characteristics of a class. They represent the data associated with objects of that class. Attributes are typically listed below the class name and are represented with a name and data type. They may also include additional information such as visibility (e.g., public, private) and default values.
- Methods: Methods define the behaviors or operations that objects of a class can perform. They represent the functions or actions associated with the class. Methods are also listed below the class name and are represented with a name, parameters (if any), return type, and additional information such as visibility and access modifiers.
- Relationships and Associations: Class diagrams depict relationships and associations between classes. Some commonly used relationships include:
- Association: It represents a relationship between two classes, indicating that objects of one class are connected to objects of another class. Associations are typically represented by a line connecting the classes, with optional multiplicities and role names.

- Inheritance/Generalization: It represents an is-a relationship, indicating that one class (subclass) inherits properties and behaviors from another class (superclass). Inheritance is depicted by an arrow pointing to the superclass.
- Aggregation and Composition: These represent has-a relationships, indicating that one class is composed of or contains other classes. Aggregation represents a weak relationship, while composition represents a stronger relationship where the lifetime of the contained class depends on the container class.
- Lifelines: Lifelines represent the lifespan of an object during the interaction. They are drawn as vertical dashed lines extending from the object's rectangle. Lifelines show the existence of an object over a period of time.
- Messages: Messages indicate the communication or interaction between objects. They are represented by arrows with labels indicating the name of the message and, optionally, any parameters or return values. Messages flow horizontally between the lifelines of objects.
- Self-Call: A self-call or self-message occurs when an object sends a message to itself. It is represented by an arrow that loops back to the same lifeline.
- Activation Bars: Activation bars, also known as execution occurrences, illustrate the time period during which an object is active and processing a message. They are depicted as horizontal bars on the lifeline.
- Lifelines: Lifelines represent the lifespan of an object during the interaction. They are drawn as vertical dashed lines extending from the object's rectangle. Lifelines show the existence of an object over a period of time.
- Messages: Messages indicate the communication or interaction between objects. They are represented by arrows with labels indicating the name of the message and, optionally, any parameters or return values. Messages flow horizontally between the lifelines of objects.
- Self-Call: A self-call or self-message occurs when an object sends a message to itself. It is represented by an arrow that loops back to the same lifeline.
- Activation Bars: Activation bars, also known as execution occurrences, illustrate the time period during which an object is active and processing a message. They are depicted as horizontal bars on the lifeline.



. **Fig.5.2.2 Class diagram**

Activity diagram:

The process flows in the system are captured in the activity diagram. Similar to a state diagram, an activity diagram also consists of activities, actions, transitions, initial and final states, and guard conditions.

- Initial Node: It represents the starting point of the activity diagram. It is depicted as a filled circle.
- Activity: An activity represents a specific action or task performed within the system. It represents a unit of work or behavior that can be performed sequentially or in parallel. Activities are depicted as rounded rectangles.
- Decision Node: It represents a point in the diagram where a decision is made, leading to different paths based on specified conditions. It is depicted as a diamond shape.
- Control Flow: Control flows represent the sequence of activities or actions in the diagram. They show the flow of control from one activity to another, indicating the order in which the activities are performed. Control flows are represented by arrows connecting the activity nodes.
- Merge Node: It represents a point where multiple control flows converge into a single path. It is used to synchronize the flow after parallel activities. Merge nodes are depicted as a diamond shape with multiple incoming flows and a single outgoing flow.
- Lifelines: Lifelines represent the lifespan of an object during the interaction. They are drawn as vertical dashed lines extending from the object's rectangle. Lifelines show the existence of an object over a period of time.
- Messages: Messages indicate the communication or interaction between objects. They are represented by arrows with labels indicating the name of the message and, optionally, any parameters or return values. Messages flow horizontally between the lifelines of objects.
- Self-Call: A self-call or self-message occurs when an object sends a message to itself. It is represented by an arrow that loops back to the same lifeline.
- Activation Bars: Activation bars, also known as execution occurrences, illustrate the time period during which an object is active and processing a message. They are depicted as horizontal bars on the lifeline.

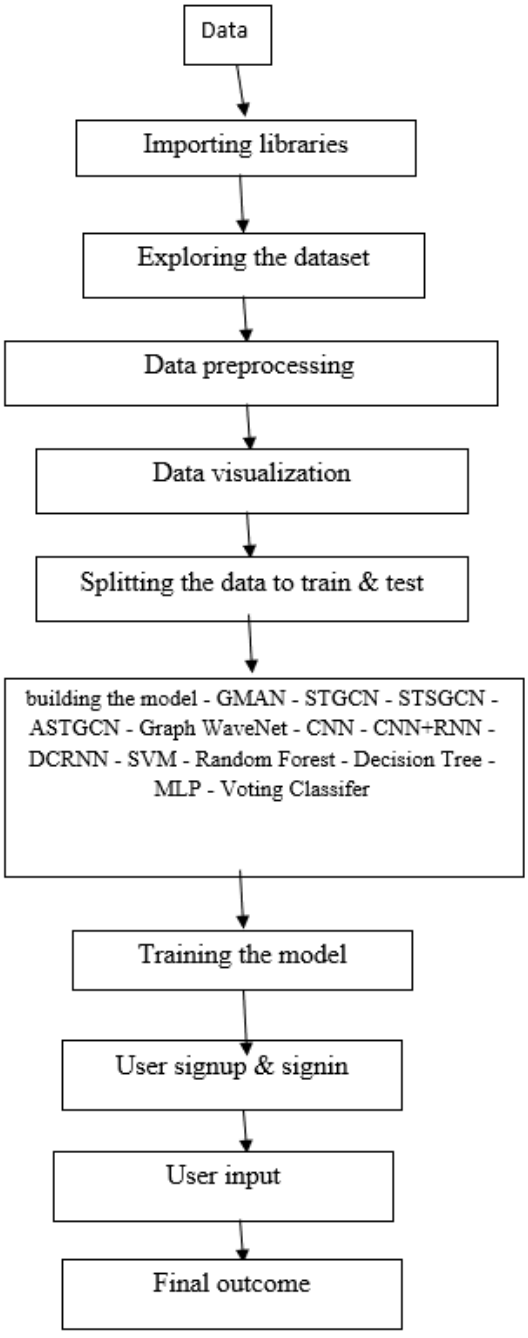


Fig.5.2.3 Activity diagram

Sequence diagram:

A sequence diagram represents the interaction between different objects in the system. The important aspect of a sequence diagram is that it is time-ordered. This means that the exact sequence of the interactions between the objects is represented step by step. Different objects in the sequence diagram interact with each other by passing "messages".

- Lifeline: A lifeline represents an object or component participating in the sequence of interactions. It represents an instance of a class or a component in the system. Lifelines are depicted as vertical lines, extending across the diagram's timeline.
- Activation Box: An activation box represents the period of time during which an object is actively processing or executing an operation. It is drawn as a box on the lifeline, indicating when an object is performing a particular action or method.
- Messages: Messages represent the communication or interaction between lifelines. They indicate the flow of information or the invocation of methods between objects. Messages are depicted as arrows, labeled with the name of the method or the information being passed.
- Return Messages: Return messages depict the response or return values sent back from an object to the sender. They represent the completion of a method or operation and are usually shown as a dashed line with an arrow.
- Self-Invocation: Self-Invocation occurs when an object invokes a method or performs an action on itself. It is depicted by a loopback arrow
- Combined Fragments: Combined fragments represent alternative or conditional sequences of interactions. They allow for representing conditions, loops, or parallel execution within the sequence diagram. Common combined fragments include alt (alternative), opt (optional), loop (repetition), and par (parallel execution).
- Sequence diagrams are widely used in software development to model the interactions between objects or components during the execution of a system or a specific scenario. They help visualize the order of method invocations, the flow of control, and the collaborations between objects.

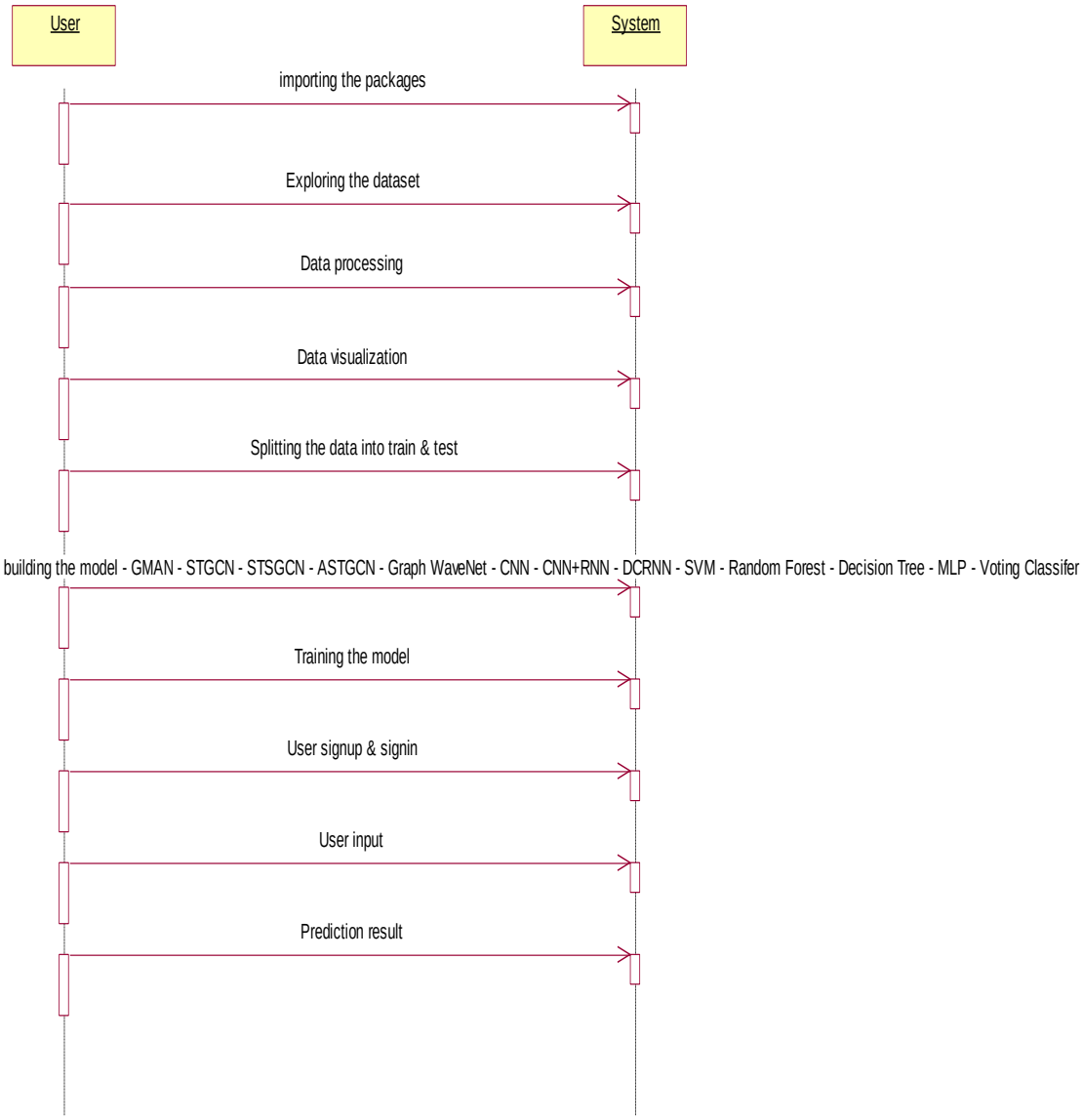


Fig.5.2.4 Sequence diagram

Collaboration diagram:

A collaboration diagram groups together the interactions between different objects. The interactions are listed as numbered interactions that help to trace the sequence of the interactions. The collaboration diagram helps to identify all the possible interactions that each object has with other objects.

Here are the key elements and features of a collaboration diagram:

- Objects: In a collaboration diagram, objects or instances of classes are depicted as rectangles with the object name written inside. Each object represents an entity that plays a role in the system.
- Lifelines: Lifelines represent the lifespan of an object during the interaction. They are drawn as vertical dashed lines extending from the object's rectangle. Lifelines show the existence of an object over a period of time.
- Messages: Messages indicate the communication or interaction between objects. They are represented by arrows with labels indicating the name of the message and, optionally, any parameters or return values. Messages flow horizontally between the lifelines of objects.
- Self-Call: A self-call or self-message occurs when an object sends a message to itself. It is represented by an arrow that loops back to the same lifeline.
- Activation Bars: Activation bars, also known as execution occurrences, illustrate the time period during which an object is active and processing a message. They are depicted as horizontal bars on the lifeline.
- Collaboration Links: Collaboration links, shown as dotted lines, represent relationships or associations between objects participating in the interaction. They can indicate the flow of messages, dependencies, or any other connections between objects.
- Constraints: Constraints or conditions can be specified on messages to indicate certain conditions that must be satisfied for the message to be sent or received.

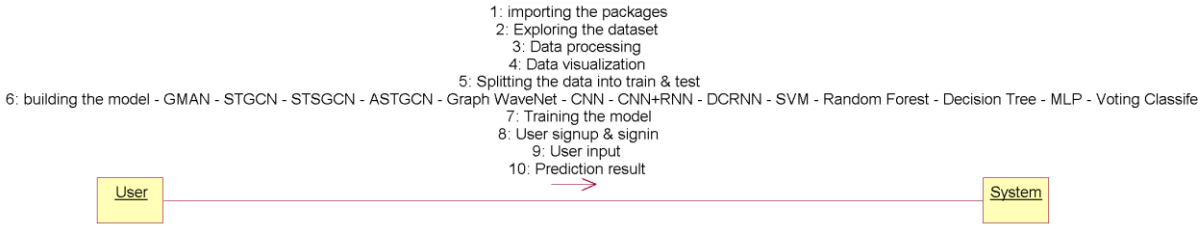


Fig.5.2.5 Collaboration diagram

Component diagram:

The component diagram represents the high-level parts that make up the system. This diagram depicts, at a high level, what components form part of the system and how they are interrelated. A component diagram depicts the components culled after the system has undergone the development or construction phase.

In a component diagram, the primary elements are:

- Component: A component represents a modular, self-contained, and replaceable part of a system. It encapsulates the implementation and provides a well-defined interface for interacting with other components. Components are depicted as rectangles with the component name inside.
- Interface: An interface represents a contract or specification that defines the services or operations provided by a component. It describes the public methods, attributes, and dependencies required to interact with the component. Interfaces are depicted as circles labeled with the interface name.
- Dependency Relationship: A dependency relationship indicates that one component depends on another component or interface. It represents a uses or requires relationship between components. Dependency relationships are depicted by arrows pointing from the dependent component to the component it depends on.
- Association Relationship: An association relationship represents a connection or relationship between two components. It indicates that one component is associated with another component, but the dependency is not as strong as in a dependency

relationship. Association relationships are depicted by lines connecting the components, with optional role names and multiplicities.

- Provided and Required Interfaces: Provided interfaces indicate the interfaces that a component offers or implements. They are depicted as small circles attached to the component. Required interfaces represent the interfaces that a component requires or depends on. They are depicted as small circles connected to the component by a dashed line.
- Package: A package is used to group related components together. It represents a namespace or container for organizing the components in the system. Packages are depicted as rectangles with the package name inside.

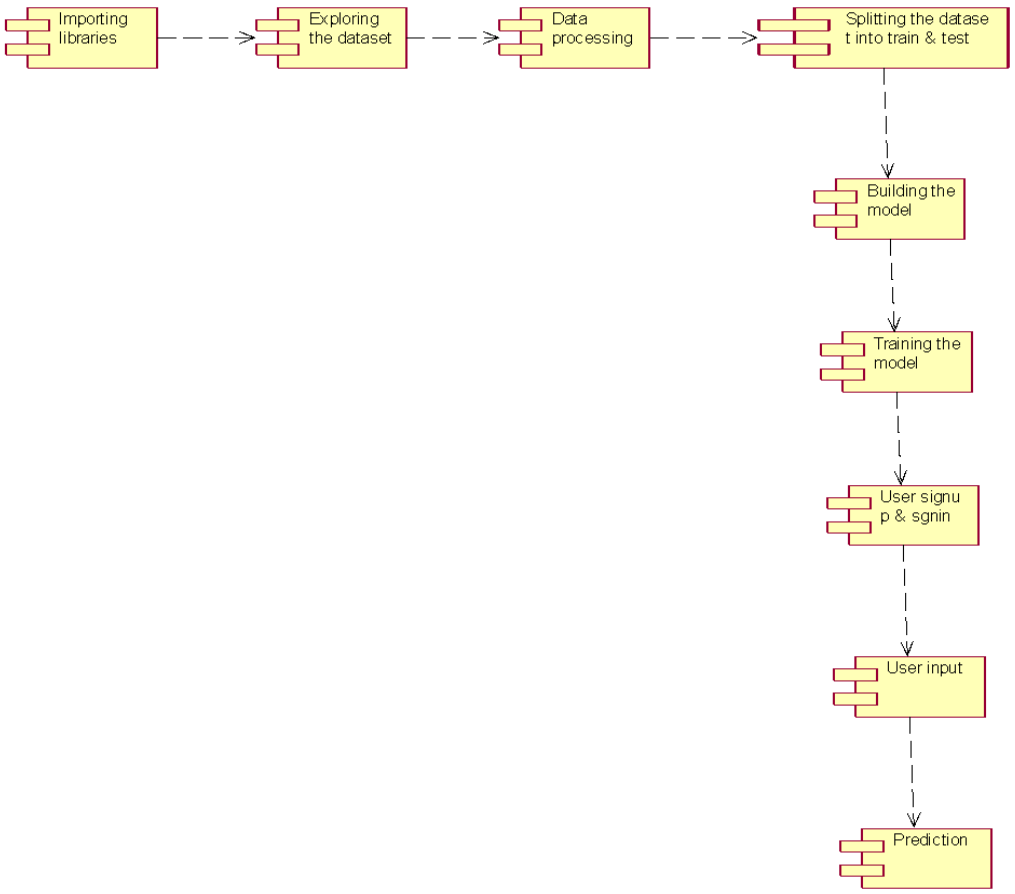


Fig.5.2.6 Component diagram

Deployment diagram:

The deployment diagram captures the configuration of the runtime elements of the application. This diagram is by far most useful when a system is built and ready to be deployed.

In a deployment diagram, the primary elements are:

- Node: A node represents a physical device or environment where software components can be deployed. It could be a server, a desktop computer, a mobile device, or any other hardware entity. Nodes are depicted as boxes or rectangles.
- Artifact: An artifact represents a tangible piece of software that is deployed on a node. It could be a component, module, file, executable, or any other software entity. Artifacts are depicted as rectangles or squares and are placed within the nodes to represent their deployment.
- Deployment Relationship: A deployment relationship indicates the mapping between an artifact and a node. It shows that a specific artifact is deployed on a particular node. Deployment relationships are represented by lines connecting artifacs to nodes.
- Communication Path: A communication path represents the communication or interaction between nodes. It indicates how nodes exchange data or communicate with each other. Communication paths are depicted as lines connecting the nodes, and they may be labeled to indicate the communication protocol or technology used.
- Stereotypes and Annotations: Deployment diagrams may use stereotypes and annotations to provide additional information or clarify the meaning of elements. Stereotypes, represented by << >>, can be used to indicate specific types of nodes or artifacts, such as web servers or databases. Annotations can be used to provide descriptive information or notes about specific elements in the diagram.
- Decision Node: It represents a point in the diagram where a decision is made, leading to different paths based on specified conditions. It is depicted as a diamond shape.
- Control Flow: Control flows represent the sequence of activities or actions in the diagram. They show the flow of control from one activity to another, indicating the

order in which the activities are performed. Control flows are represented by arrows connecting the activity nodes.

- Merge Node: It represents a point where multiple control flows converge into a single path. It is used to synchronize the flow after parallel activities. Merge nodes are depicted as a diamond shape with multiple incoming flows and a single outgoing flow.
- Lifelines: Lifelines represent the lifespan of an object during the interaction. They are drawn as vertical dashed lines extending from the object's rectangle. Lifelines show the existence of an object over a period of time.

Fig.5.2.7 Deployment diagram

CHAPTER 6

IMPLEMENTATION

MODULES:

1. importing the packages: using this module we will import all packages
2. exploring the dataset - Arrhythmia Data: Using this module we will upload dataset
3. data processing: Using this module we will read data for processing
4. visualization using seaborn & matplotlib: Using this module will get graphical representation of information and data.
5. Splitting the data to train and test: Using this module will divide dataset into train & test for processing
6. building the model: Using this module we will build all algorithms
   - GMAN
   - STGCN
   - STSGCN
   - ASTGCN
   - Graph WaveNet
   - CNN
   - CNN+RNN
   - DCRNN
   - SVM
   - Random Forest
   - Decision Tree
   - MLP
   - Voting Classifer
7. training the model: Using this module algorithms trained for processing & prediction building the model with Voting Classifier since it gives better accuracy comparing with Other Models
8. Flask Framework with Sqlite for signup and signin: Using this module user will get register & login

importing the packages

9. User gives input as Feature Values : Uisng this module user gives input for prediction the given input is preprocessed for prediction

10. trained model is used for prediction: Using this module predicted result displayed final outcome is displayed through frontend

Note:

Extension - Voting Classifier, MLP, CNN, CNN+RNN model is build as Extension for feature values

used for prediction since where the extension model Voting Classifier gives better accuracy of 100% comparing with other Models

ALGORITHMS:

- Random Forest: Random forest is a Supervised Machine Learning Algorithm that is used widely in Classification and Regression problems. It builds decision trees on different samples and takes their majority vote for classification and average in case of regression.
- Decision Tree: Decision trees use multiple algorithms to decide to split a node into two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. In other words, we can say that the purity of the node increases with respect to the target variable.
- MLP: A multilayer perceptron (MLP) is a fully connected class of feedforward artificial neural network (ANN). The term MLP is used ambiguously, sometimes loosely to mean any feedforward ANN, sometimes strictly to refer to networks composed of multiple layers of perceptrons (with threshold activation); see § Terminology. Multilayer perceptrons are sometimes colloquially referred to as "vanilla" neural networks, especially when they have a single hidden layer.
- Support Vector Machine: Support Vector Machine(SVM) is a supervised machine learning algorithm used for both classification and regression. Though we say regression problems as well its best suited for classification. The objective of SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points.

- Voting Classifier: A voting classifier is a machine learning estimator that trains various base models or estimators and predicts on the basis of aggregating the findings of each base estimator. The aggregating criteria can be combined decision of voting for each estimator output.
- BF Tree: The breadth-first search or BFS algorithm is used to search a tree or graph data structure for a node that meets a set of criteria. It begins at the root of the tree or graph and investigates all nodes at the current depth level before moving on to nodes at the next depth level.
- BayesNet: Bayesian networks are a type of Probabilistic Graphical Model that can be used to build models from data and/or expert opinion. They can be used for a wide range of tasks including prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction and decision making under uncertainty.
- CNN: A CNN is a kind of network architecture for deep learning algorithms and is specifically used for image recognition and tasks that involve the processing of pixel data. There are other types of neural networks in deep learning, but for identifying and recognizing objects, CNNs are the network architecture of choice.

6.2 SAMPLE CODE:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
```

```
from sklearn.metrics import
classification_report,accuracy_score,confusion_matrix
from sklearn.metrics import
auc,roc_auc_score,roc_curve,precision_score,recall_score,f1_score
import time as timer
from sklearn.inspection import permutation_importance
# import package
# open dataset
filename = "train_2v.csv"
data = pd.read_csv(filename)
with pd.option_context('expand_frame_repr', False):
    print(data.head())
print("Data shape: {}".format(data.shape))
miss_val = data.isnull().sum()/len(data)*100
print(miss_val)
print("# Missing values in variable bmi\t\t: {:.2f}%".format(miss_val['bmi']))
print("# Missing values in variable smoking_status\t:
{:.2f}%".format(miss_val['smoking_status']))
print("Data shape: {}".format(data.shape))
# Safely disable new warning with the chained assignment.
pd.options.mode.chained_assignment = None  # default='warn'
# replace missing values in variable 'bmi' with its mean
data['bmi']=data['bmi'].fillna(data['bmi'].mean())
# remove (drop) data associated with missing values in variable 'smoking_status'
clean_data = data[data['smoking_status'].notnull()]
# drop variable 'id'
clean_data.drop(columns='id',axis=1,inplace=True)
# validate there's no more missing values
miss_val = clean_data.isnull().sum()/len(clean_data)*100
print(miss_val)
print("# Missing values in variable 'bmi'\t\t: {}".format(miss_val['bmi']))
```

```
print("# Missing values in variable 'smoking_status'\t:
{}".format(miss_val['smoking_status']))
print("Shape of data without missing values: {}".format(clean_data.shape))
print("Unique 'gender': {}".format(clean_data['gender'].unique()))
print("Unique 'ever_married': {}".format(clean_data['ever_married'].unique()))
print("Unique 'work_type': {}".format(clean_data['work_type'].unique()))
print("Unique 'Residence_type':
{}".format(clean_data['Residence_type'].unique()))
print("Unique 'smoking_status':
{}".format(clean_data['smoking_status'].unique()))
# create encoder for each categorical variable
label_gender = LabelEncoder()
label_married = LabelEncoder()
label_work = LabelEncoder()
label_residence = LabelEncoder()
label_smoking = LabelEncoder()
clean_data['gender'] = label_gender.fit_transform(clean_data['gender'])
clean_data['ever_married'] =
label_married.fit_transform(clean_data['ever_married'])
clean_data['work_type']= label_work.fit_transform(clean_data['work_type'])
clean_data['Residence_type']=
label_residence.fit_transform(clean_data['Residence_type'])
clean_data['smoking_status']=
label_smoking.fit_transform(clean_data['smoking_status'])
with pd.option_context('expand_frame_repr', False):
    print(clean_data.head())
fig, ax = plt.subplots(figsize=(8,6))
im = ax.matshow(clean_data.corr())
ax.set_xticks(np.arange(clean_data.shape[1]))
ax.set_yticks(np.arange(clean_data.shape[1]))
ax.set_xticklabels(clean_data.columns,rotation=90)
```

```
ax.set_yticklabels(clean_data.columns)
# Create colorbar
cbar = ax.figure.colorbar(im, ax=ax)
cbar.ax.set_ylabel("Correlation", rotation=-90, va="bottom", fontsize=12)
fig.tight_layout()
plt.show()
fig = clean_data.hist(figsize=(10,8))
plt.tight_layout()
plt.show()
print("Classification report for SVM:
\n{}".format(classification_report(y_test,y_svm)))
print("Confusion matrix for SVM:
\n{}".format(confusion_matrix(y_test,y_svm)))
print("Accuracy score for SVM: {:.2f}".format(accuracy_score(y_test,y_svm)))
# calculate precision, recall, and f1 scores
prec_svm = precision_score(y_test,y_svm)
rec_svm = recall_score(y_test,y_svm)
f1_svm = f1_score(y_test,y_svm)
print("Precision score for SVM: {:.2f}".format(prec_svm))
print("Recall score for SVM: {:.2f}".format(rec_svm))
print("F1 score for SVM: {:.2f}".format(f1_svm))
# calculate sensitivity, specificity, and auc
sens_svm,spec_svm = calc_sens_spec(y_test,y_svm)
fpr, tpr, _ = roc_curve(y_test,  y_svm_prob[:,1])
auc_svm = roc_auc_score(y_test, y_svm_prob[:,1])
print("Sensitivity score for SVM: {:.2f}".format(sens_svm))
print("Specitivity score for SVM: {:.2f}".format(spec_svm))
print("AUC score for SVM: {:.2f}".format(auc_svm))
fig, ax = plt.subplots()
ax.plot(fpr, tpr, color='blue', label='ROC curve (area = %0.2f)' % auc_svm)
ax.plot([0, 1], [0, 1], color='green', linestyle='--')
```

```
ax.set_xlim([-0.05, 1.0])
ax.set_ylim([0.0, 1.05])
ax.set_xlabel('False Positive Rate')
ax.set_ylabel('True Positive Rate')
ax.set_title('Receiver Operating Characteristic (SVM)')
ax.legend(loc="lower right")
plt.show()
```

CHAPTER 7

SOFTWARE ENVIRONMENT

PYTHON LANGUAGE:

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed. Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

Python is a dynamic, high-level, free open source, and interpreted programming language. It supports object-oriented programming as well as procedural-oriented programming. In Python, we don’t need to declare the type of variable because it is a dynamically typed language. For example, x = 10 Here, x can be anything such as String, int, etc.

Features in Python:

There are many features in Python, some of which are discussed below as follows:

1. Free and Open Source

Python language is freely available at the official website and you can download it from the given download link below click on the Download Python keyword. Download Python Since it is open-source, this means that source code is also available to the public. So you can download it, use it as well as share it.

2. Easy to code

Python is a high-level programming language. Python is very easy to learn the language as compared to other languages like C, C#, Javascript, Java, etc. It is very easy to code in the Python language and anybody can learn Python basics in a few hours or days. It is also a developer-friendly language.

3. Easy to Read

As you will see, learning Python is quite simple. As was already established, Python's syntax is really straightforward. The code block is defined by the indentations rather than by semicolons or brackets.

4. Object-Oriented Language

One of the key features of Python is Object-Oriented programming. Python supports object-oriented language and concepts of classes, object encapsulation, etc.

5. GUI Programming Support

Graphical User interfaces can be made using a module such as PyQt5, PyQt4, wxPython, or Tk in python. PyQt5 is the most popular option for creating graphical apps with Python.

6. High-Level Language

Python is a high-level language. When we write programs in Python, we do not need to remember the system architecture, nor do we need to manage the memory.

7. Extensible feature

Python is an Extensible language. We can write some Python code into C or C++ language and also we can compile that code in C/C++ language.

8. Easy to Debug

Excellent information for mistake tracing. You will be able to quickly identify and correct the majority of your program's issues once you understand how to interpret Python's error traces. Simply by glancing at the code, you can determine what it is designed to perform.

9. Python is a Portable language

Python language is also a portable language. For example, if we have Python code for windows and if we want to run this code on other platforms such as Linux, Unix, and Mac then we do not need to change it, we can run this code on any platform.

10. Python is an Integrated language

Python is also an Integrated language because we can easily integrate Python with other languages like C, C++, etc.

11. Interpreted Language:

Python is an Interpreted Language because Python code is executed line by line at a time. like other languages C, C++, Java, etc. there is no need to compile Python code this makes it easier to debug our code. The source code of Python is converted into an immediate form called bytecode.

12. Large Standard Library

Python has a large standard library that provides a rich set of modules and functions so you do not have to write your own code for every single thing. There are many libraries present in Python such as regular expressions, unit-testing, web browsers, etc.

13. Dynamically Typed Language

Python is a dynamically-typed language. That means the type (for example- int, double, long, etc.) for a variable is decided at run time not in advance because of this feature we don't need to specify the type of variable.

14. Frontend and backend development

With a new project py script, you can run and write Python codes in HTML with the help of some simple tags <py-script>, <py-env>, etc. This will help you do frontend development work in Python like javascript. Backend is the strong forte of Python it's extensively used for this work cause of its frameworks like Django and Flask.

15. Allocating Memory Dynamically

I n Python, the variable data type does not need to be specified. The memory is automatically allocated to a variable at runtime when it is given a value. Developers do not need to write int y = 18 if the integer value 15 is set to y. You may just type y=18.

LIBRARIES/PACKGES :-

Tensorflow

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open-source license on November 9, 2015.

Numpy

Numpy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined using Numpy which allows Numpy to seamlessly and speedily integrate with a wide variety of databases.

Pandas

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data load, prepare, manipulate, model, and analyze. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter Notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

Scikit – learn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use.

CHAPTER 8

SYSTEM TESTING

System testing, also referred to as system-level tests or system-integration testing, is the process in which a quality assurance (QA) team evaluates how the various components of an application interact together in the full, integrated system or application. System testing verifies that an application performs tasks as designed. This step, a kind of black box testing, focuses on the functionality of an application. System testing, for example, might check that every kind of user input produces the intended output across the application.

Phases of system testing:

A video tutorial about this test level. System testing examines every component of an application to make sure that they work as a complete and unified whole. A QA team typically conducts system testing after it checks individual modules with functional or user-story testing and then each component through integration testing.

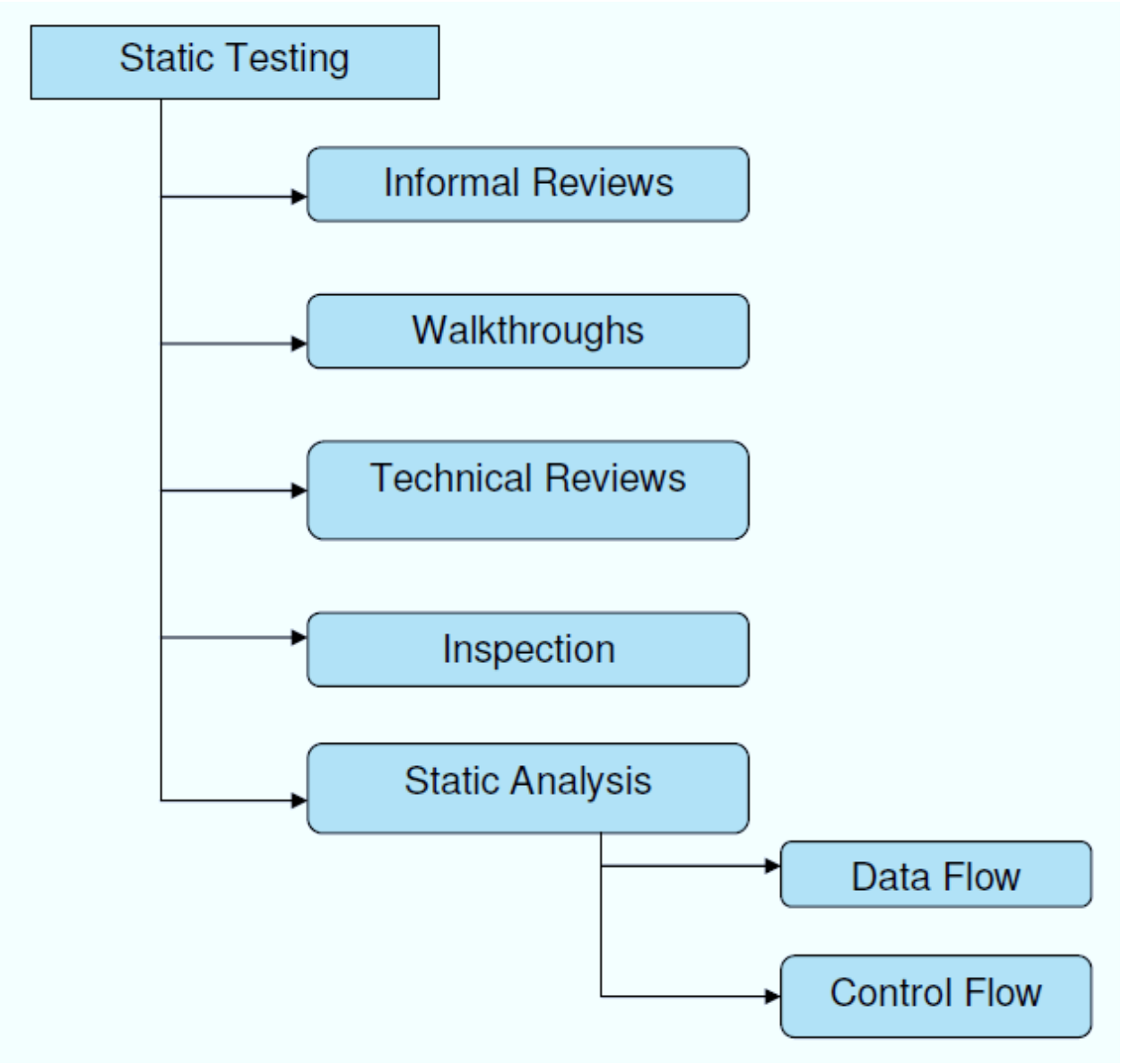
If a software build achieves the desired results in system testing, it gets a final check via acceptance testing before it goes to production, where users consume the software. An app-dev team logs all defects, and establishes what kinds and amount of defects are tolerable.

8.1Software Testing Strategies:

Optimization of the approach to testing in software engineering is the best way to make it effective. A software testing strategy defines what, when, and how to do whatever is necessary to make an end-product of high quality. Usually, the following software testing strategies and their combinations are used to achieve this major objective:

Static Testing:

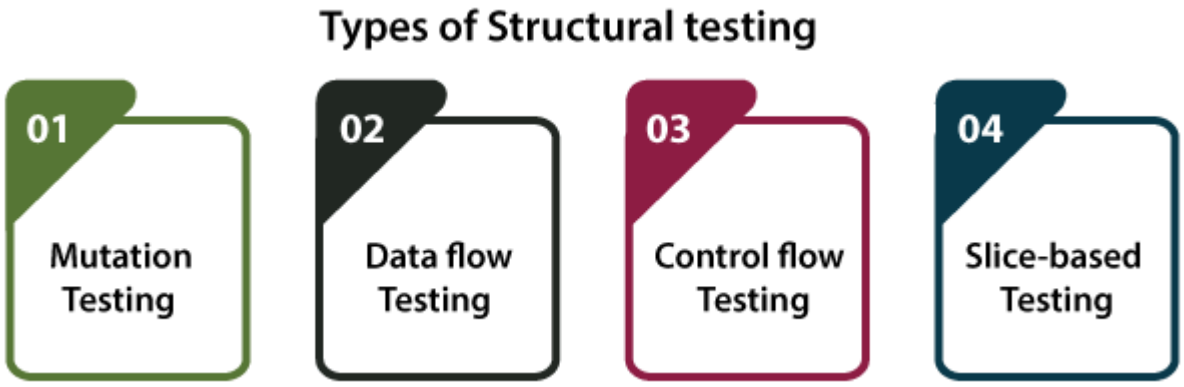
The early-stage testing strategy is static testing: it is performed without actually running the developing product. Basically, such desk-checking is required to detect bugs and issues that are present in the code itself. Such a check-up is important at the pre-deployment stage as it helps avoid problems caused by errors in the code and software structure deficits.



Structural Testing:

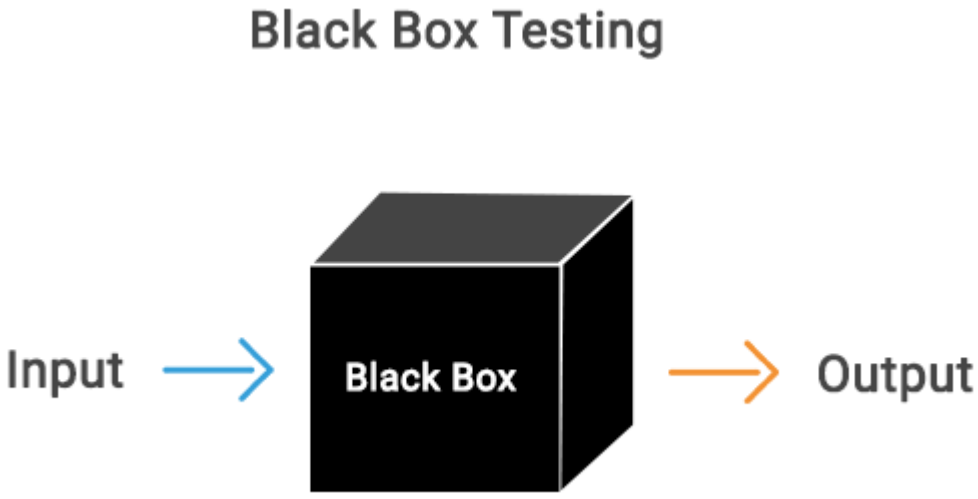
It is not possible to effectively test software without running it. Structural testing, also known as white-box testing, is required to detect and fix bugs and errors emerging during the pre-production stage of the software development process. At this stage, unit testing based on the software structure is performed using regression testing. In most cases, it is an automated process working within the test automation framework to speed up the development process at this stage. Developers and QA engineers have full access to the software's structure and data flows (data flows testing), so they could track any changes (mutation testing) in the

system's behavior by comparing the tests' outcomes with the results of previous iterations (control flow testing).



Behavioral Testing:

The final stage of testing focuses on the software's reactions to various activities rather than on the mechanisms behind these reactions. In other words, behavioral testing, also known as black-box testing, presupposes running numerous tests, mostly manual, to see the product from the user's point of view. QA engineers usually have some specific information about a business or other purposes of the software ('the black box') to run usability tests, for example, and react to bugs as regular users of the product will do. Behavioral testing also may include automation (regression tests) to eliminate human error if repetitive activities are required. For example, you may need to fill 100 registration forms on the website to see how the product copes with such an activity, so the automation of this test is preferable.



8.2 TEST CASES:

S.No	INPUT	If available	If not available
1	User signup	User get registered into the application	There is no process
2	User signin	User get login into the application	There is no process
3	Enter input for prediction	Prediction result displayed	There is no process

CHAPTER 9

SCREENS

9.1 Index page of Traffic Predictor

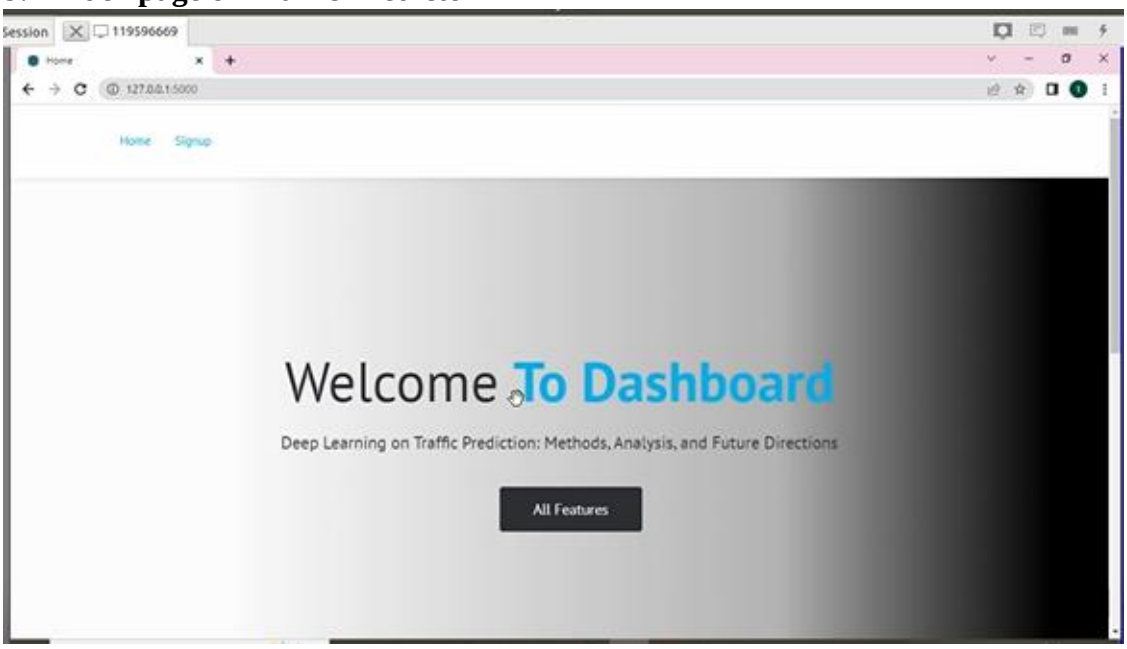


Fig 9.1: Basic Design of UI

9.2 Registration page of Traffic Predictor

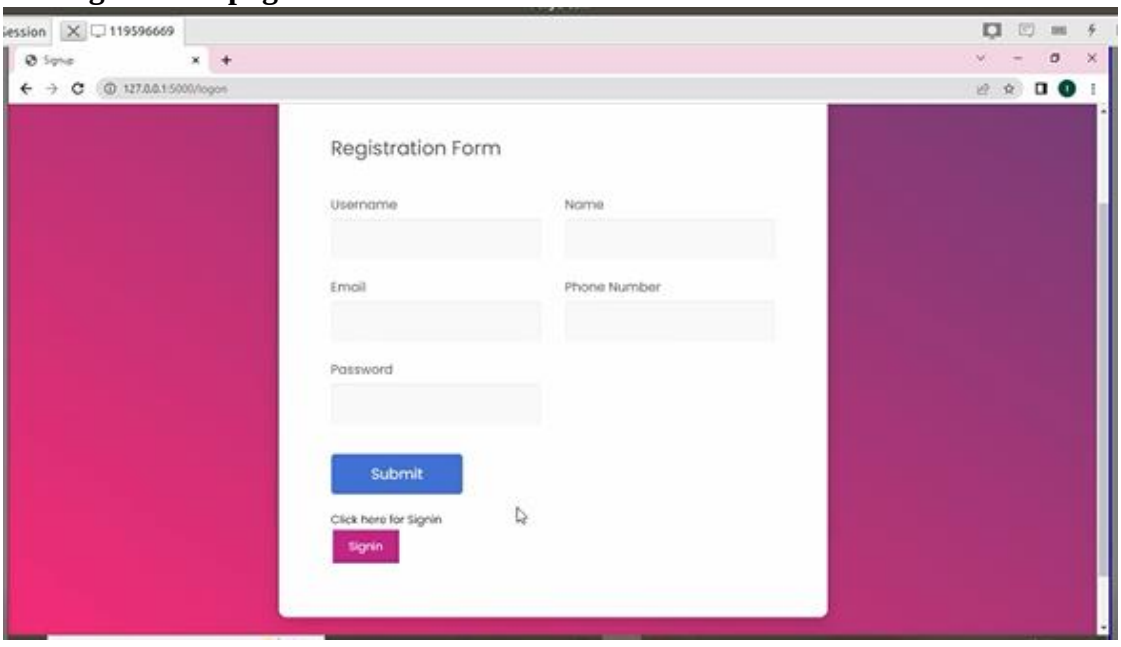


Fig 9.2: Registration page of Traffic Prediction

9.3 Login page of Traffic Predictor

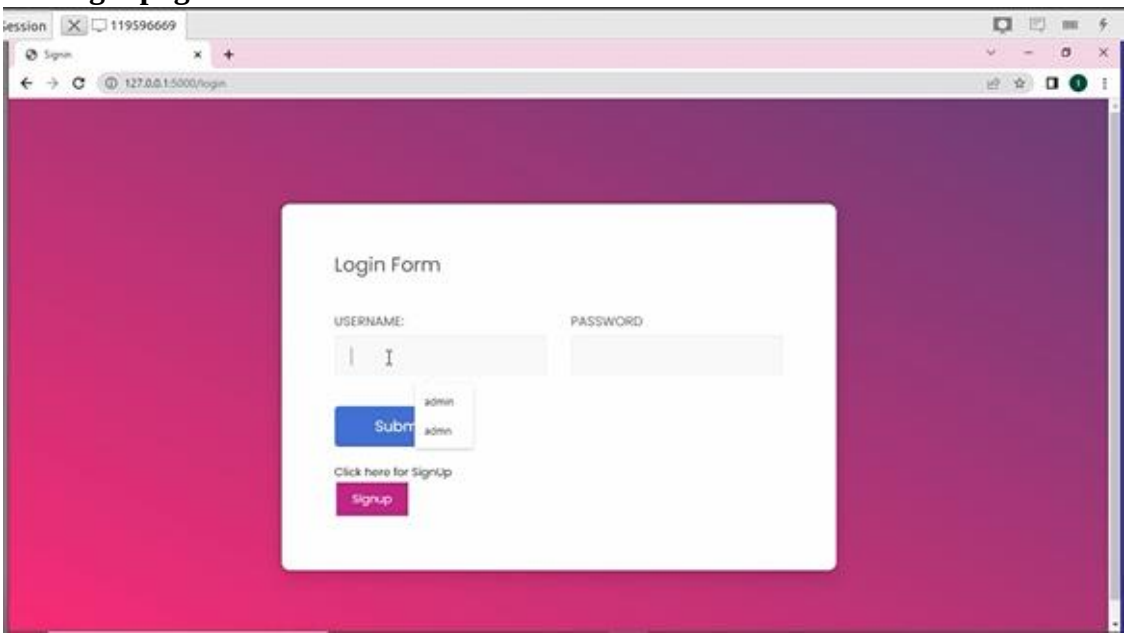


Fig 9.3: Login page of Traffic Prediction

9.4 Input page of Traffic Predictor

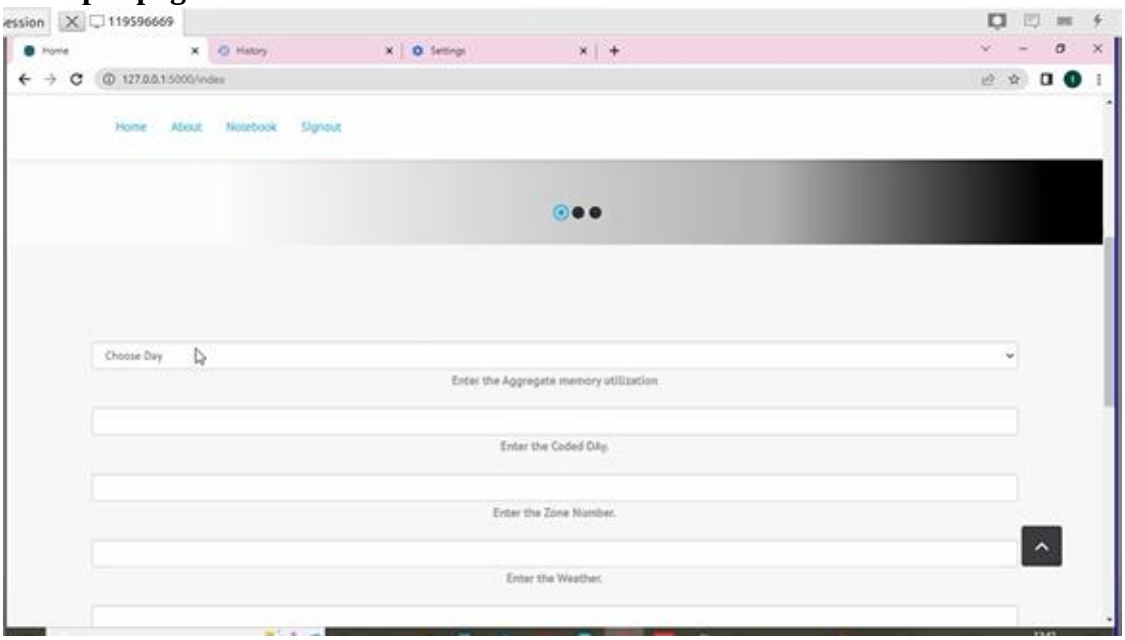


Fig 9.4: Prediction input for Traffic Prediction

On submission of this form, data the model gives the result in the form of Text; as shown in following figures

9.5 Prediction output for Moving Traffic

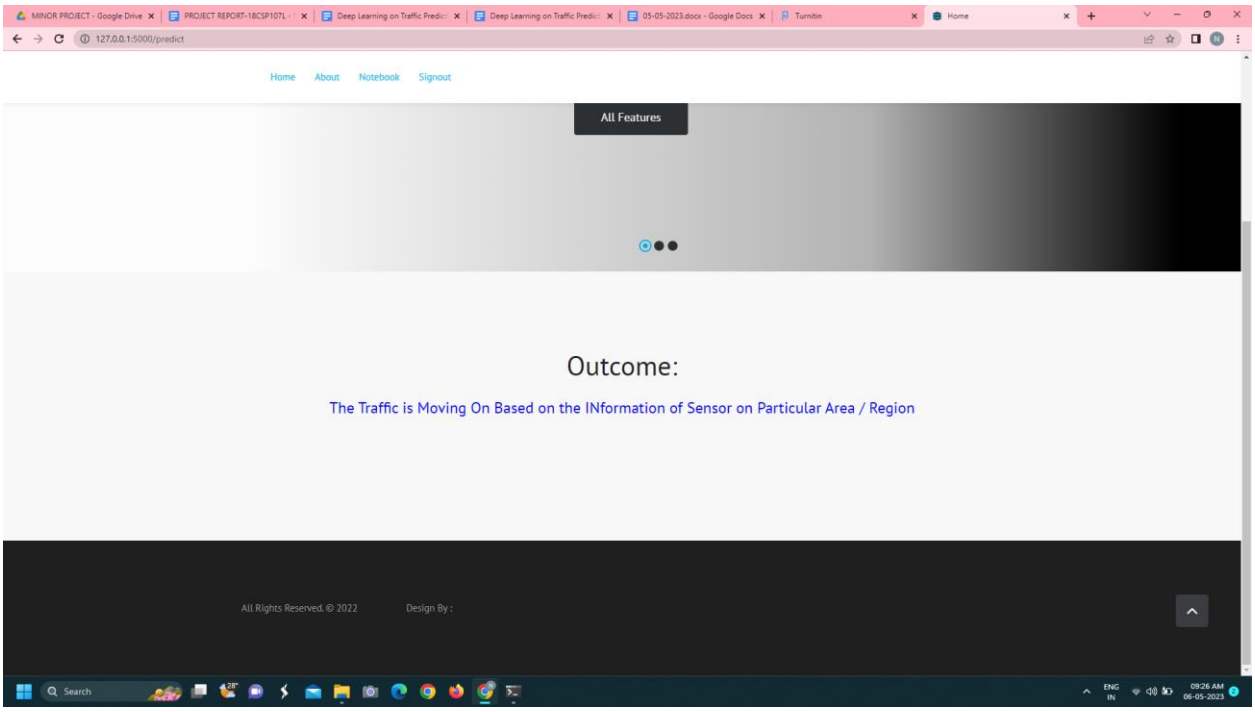


Fig 9.5: Prediction output for Moving Traffic

9.6 Prediction output for Heavy Traffic

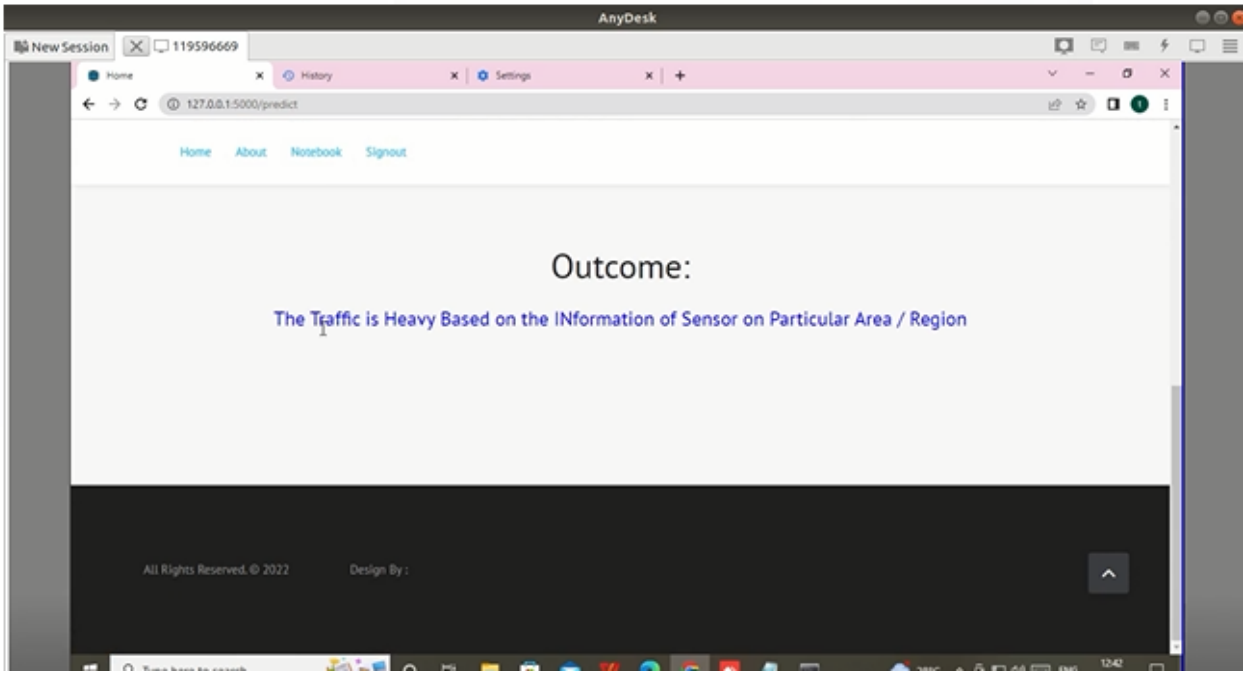


Fig 9.6: Prediction output for Heavy Traffic

CHAPTER 10

CONCLUSION

In this paper, we conduct a comprehensive survey of various deep learning architectures fot traffic prediction. More specifically, we first summarize the existing traffic prediction methods, and give a taxonomy of them. Then, we list the representative results in different traffic prediction tasks, comprehensively provide public available traffic datasets, and conduct a series of experiments to investigate the performance of existing traffic prediction methods. Finally, some major challenges and future research directions are discussed. This paper is suitable for participators to quickly understand the traffic prediction, so as to find branches they are interested in. It also provides a good reference and inquiry for researchers in this field, which can facilitate the relevant research.

CHAPTER 11

REFERENCES

[1] J. Zhang, Y. Zheng, D. Qi, R. Li, and X. Yi, “DNN-based prediction model for spatio-temporal data,” in Proc. 24th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst., Oct. 2016, pp. 1–4.

[2] E. I. Vlahogianni, M. G. Karlaftis, and J. C. Golias, “Short-term traffic forecasting: Where we are and where we’re going,” Transp. Res. C, Emerg. Technol., vol. 43, pp. 3–19, Jun. 2014.

[3] Y. Li and C. Shahabi, “A brief overview of machine learning methods for short-term traffic forecasting and future directions,” SIGSPATIAL Special, vol. 10, no. 1, pp. 3–9, Jun. 2018, doi: 10.1145/3231541.3231544.

[4] A. M. Nagy and V. Simon, “Survey on traffic prediction in smart cities,” Pervas. Mobile Comput., vol. 50, pp. 148–163, Oct. 2018.

[5] A. Singh, A. Shadan, R. Singh, and Ranjeet, “Traffic forecasting,” Int. J. Sci. Res. Rev., vol. 7, no. 3, pp. 1565–1568, 2019.

[6] A. Boukerche and J. Wang, “Machine learning-based traffic prediction models for intelligent transportation systems,” Comput. Netw., vol. 181, Nov. 2020, Art. no. 107530.

[7] I. Lana, J. D. Ser, M. Velez, and E. I. Vlahogianni, “Road traffic forecasting: Recent advances and new challenges,” IEEE Intell. Transp. Syst. Mag., vol. 10, no. 2, pp. 93–109, Apr. 2018.

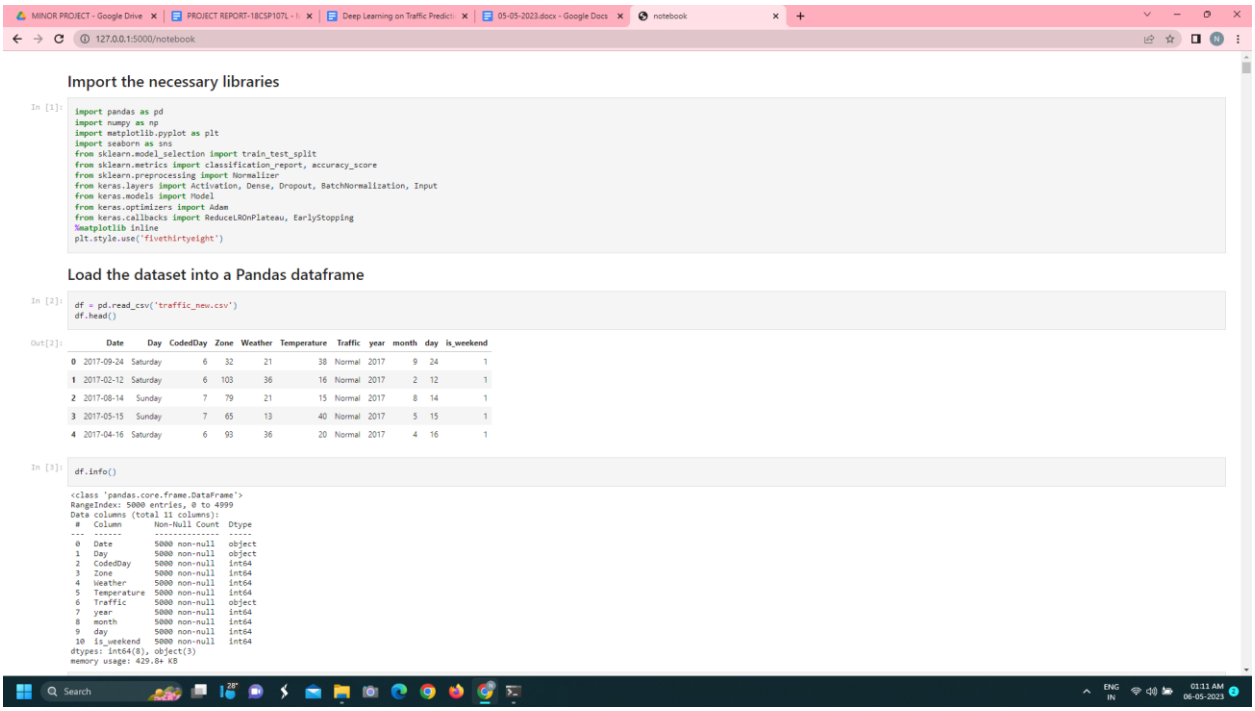
[8] D. A. Tedjopurnomo, Z. Bao, B. Zheng, F. Choudhury, and A. K. Qin, “A survey on modern deep neural network for traffic prediction: Trends, methods and challenges,” IEEE Trans. Knowl. Data Eng., early access, Jun. 9, 2020, doi: 10.1109/TKDE.2020.3001195.

[9] J. Ye, J. Zhao, K. Ye, and C. Xu, “How to build a graphbased deep learning architecture in traffic domain: A survey,” 2020, arXiv:2005.11691. [Online]. Available: http://arxiv.org/abs/2005.11691

[10] P. Xie, T. Li, J. Liu, S. Du, X. Yang, and J. Zhang, “Urban flow prediction from spatiotemporal data using machine learning: A survey,” Inf. Fusion, vol. 59, pp. 1–12, Jul. 2020, doi: 10.1016/j.inffus.2020.01.002.

`CHAPTER 12

APPENDIX



Import the necessary libraries

```
In [1]:
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
from sklearn.preprocessing import Normalizer
from keras.layers import Activation, Dense, Dropout, BatchNormalization, Input
from keras.models import Model
from keras.optimizers import Adam
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
%matplotlib inline
plt.style.use('fivethirtyeight')
```

Load the dataset into a Pandas dataframe

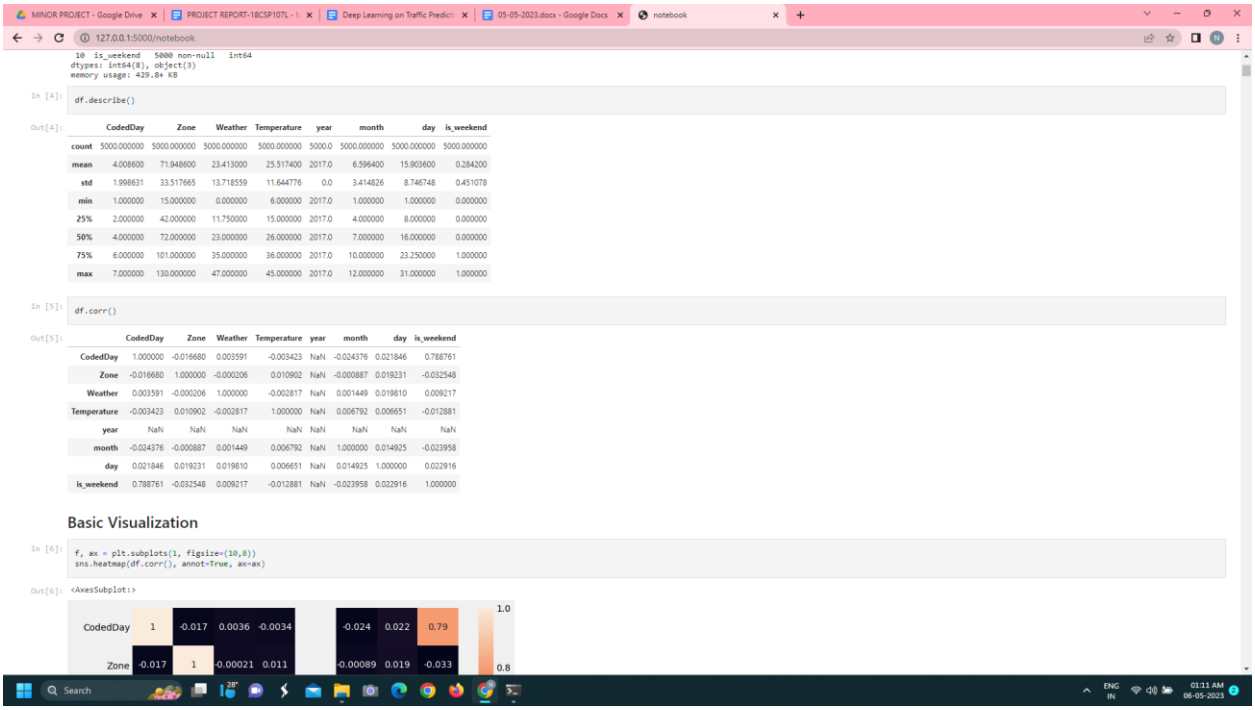
```
In [2]:
df = pd.read_csv('traffic_new.csv')
df.head()
```

Out[2]:

	Date	Day	CodedDay	Zone	Weather	Temperature	Traffic	year	month	day	is_weekend
0	2017-09-24	Saturday	6	32	21	38	Normal	2017	9	24	1
1	2017-02-12	Saturday	6	103	36	16	Normal	2017	2	12	1
2	2017-08-14	Sunday	7	79	21	15	Normal	2017	8	14	1
3	2017-05-15	Sunday	7	65	13	40	Normal	2017	5	15	1
4	2017-04-16	Saturday	6	93	36	20	Normal	2017	4	16	1

```
In [3]:
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Date         5000 non-null   object
 1   Day          5000 non-null   object
 2   CodedDay     5000 non-null   int64
 3   Zone         5000 non-null   int64
 4   Weather      5000 non-null   int64
 5   Temperature  5000 non-null   int64
 6   Traffic      5000 non-null   object
 7   year         5000 non-null   int64
 8   month        5000 non-null   int64
 9   day          5000 non-null   int64
 10  is_weekend   5000 non-null   int64
dtypes: int64(8), object(3)
memory usage: 429.8+ KB
```



```
dtypes: int64(8), object(3)
memory usage: 429.8+ KB
```

```
In [4]:
df.describe()
```

Out[4]:

	CodedDay	Zone	Weather	Temperature	year	month	day	is_weekend
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.0	5000.000000	5000.000000	5000.000000
mean	4.008600	71.948600	23.413000	25.517400	2017.0	6.596400	15.903600	0.284200
std	1.998631	33.517665	13.718559	11.644776	0.0	3.414826	8.746748	0.451078
min	1.000000	15.000000	0.000000	6.000000	2017.0	1.000000	1.000000	0.000000
25%	2.000000	42.000000	11.750000	15.000000	2017.0	4.000000	8.000000	0.000000
50%	4.000000	72.000000	23.000000	26.000000	2017.0	7.000000	16.000000	0.000000
75%	6.000000	101.000000	35.000000	36.000000	2017.0	10.000000	23.250000	1.000000
max	7.000000	130.000000	47.000000	45.000000	2017.0	12.000000	31.000000	1.000000

```
In [5]:
df.corr()
```

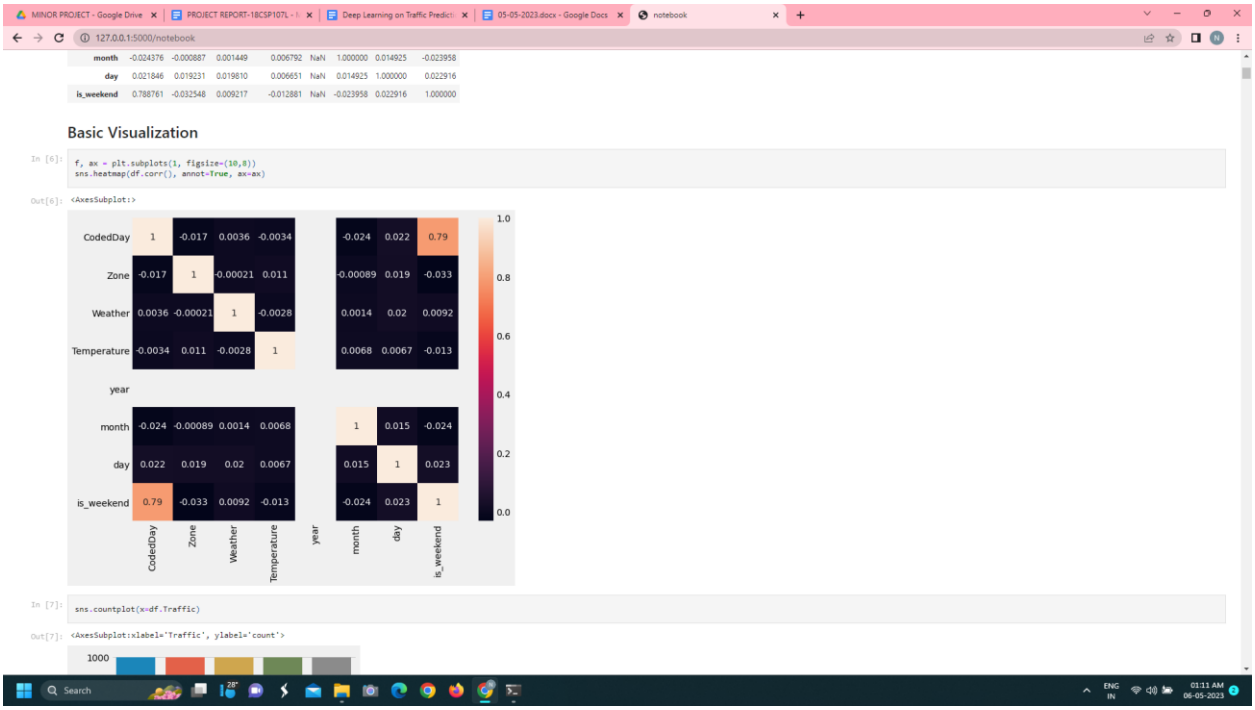
Out[5]:

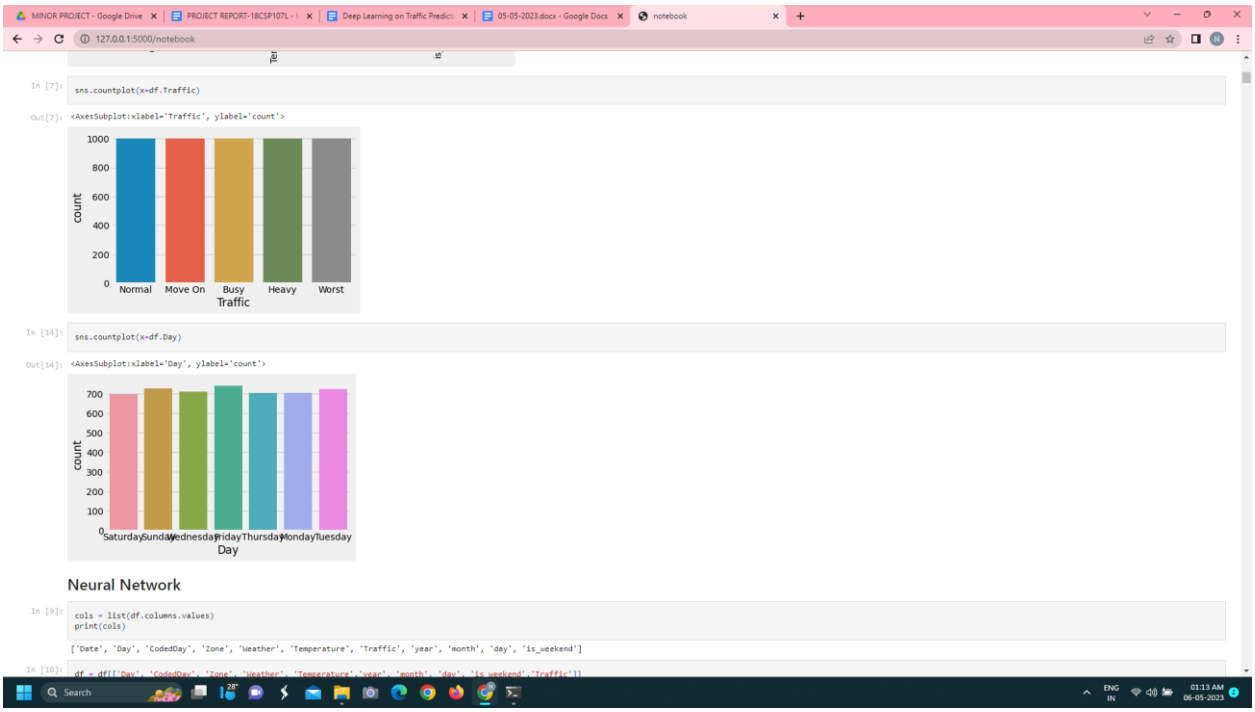
	CodedDay	Zone	Weather	Temperature	year	month	day	is_weekend
CodedDay	1.000000	-0.016680	0.003591	-0.003423	NaN	-0.024376	0.021846	0.788761
Zone	-0.016680	1.000000	-0.000206	0.010902	NaN	-0.000887	0.019231	-0.032548
Weather	0.003591	-0.000206	1.000000	-0.002817	NaN	0.001449	0.019810	0.009217
Temperature	-0.003423	0.010902	-0.002817	1.000000	NaN	0.006792	0.006651	-0.012881
year	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
month	-0.024376	-0.000887	0.001449	0.006792	NaN	1.000000	0.014925	-0.023958
day	0.021846	0.019231	0.019810	0.006651	NaN	0.014925	1.000000	0.022916
is_weekend	0.788761	-0.032548	0.009217	-0.012881	NaN	-0.023958	0.022916	1.000000

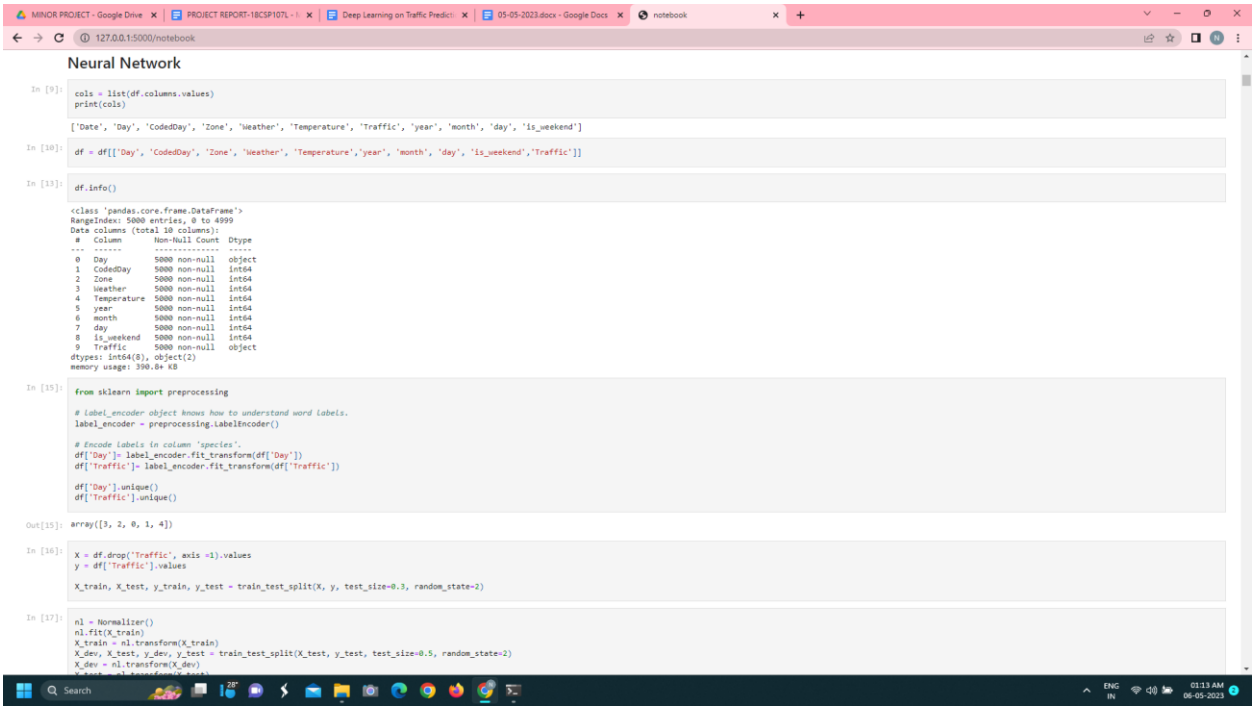
Basic Visualization

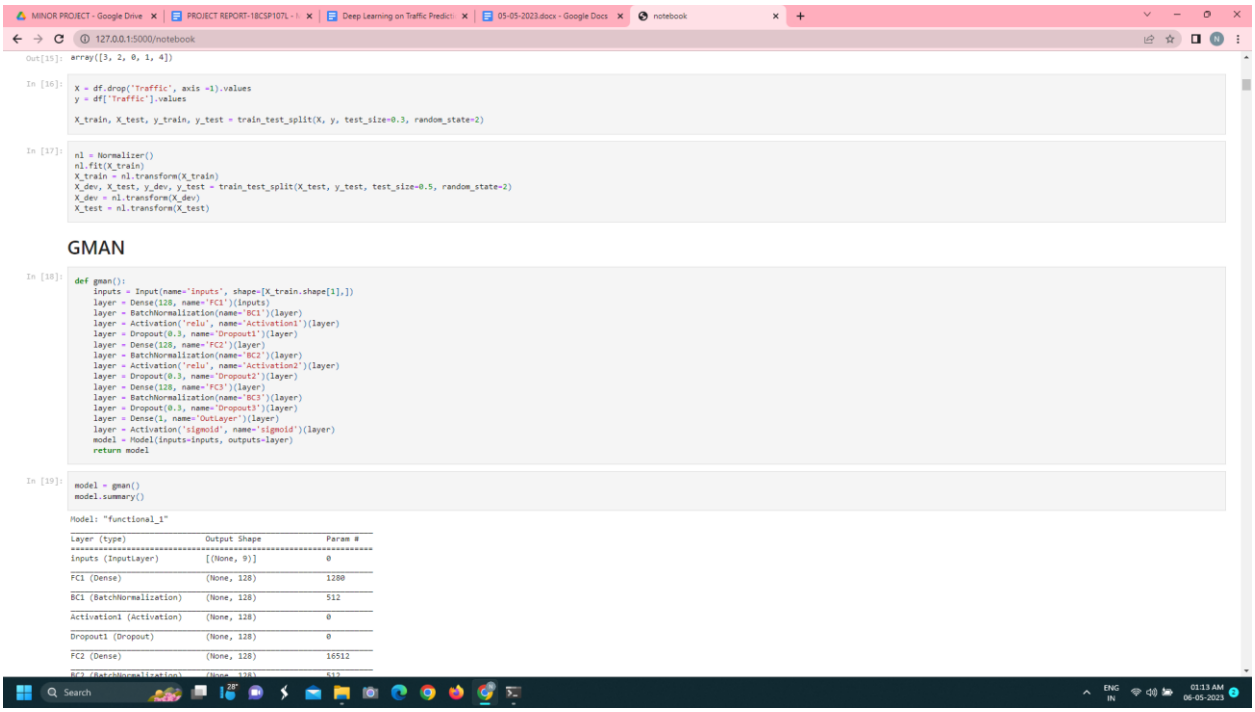
```
In [6]:
f, ax = plt.subplots(1, figsize=(10,8))
sns.heatmap(df.corr(), annot=True, ax=ax)
```

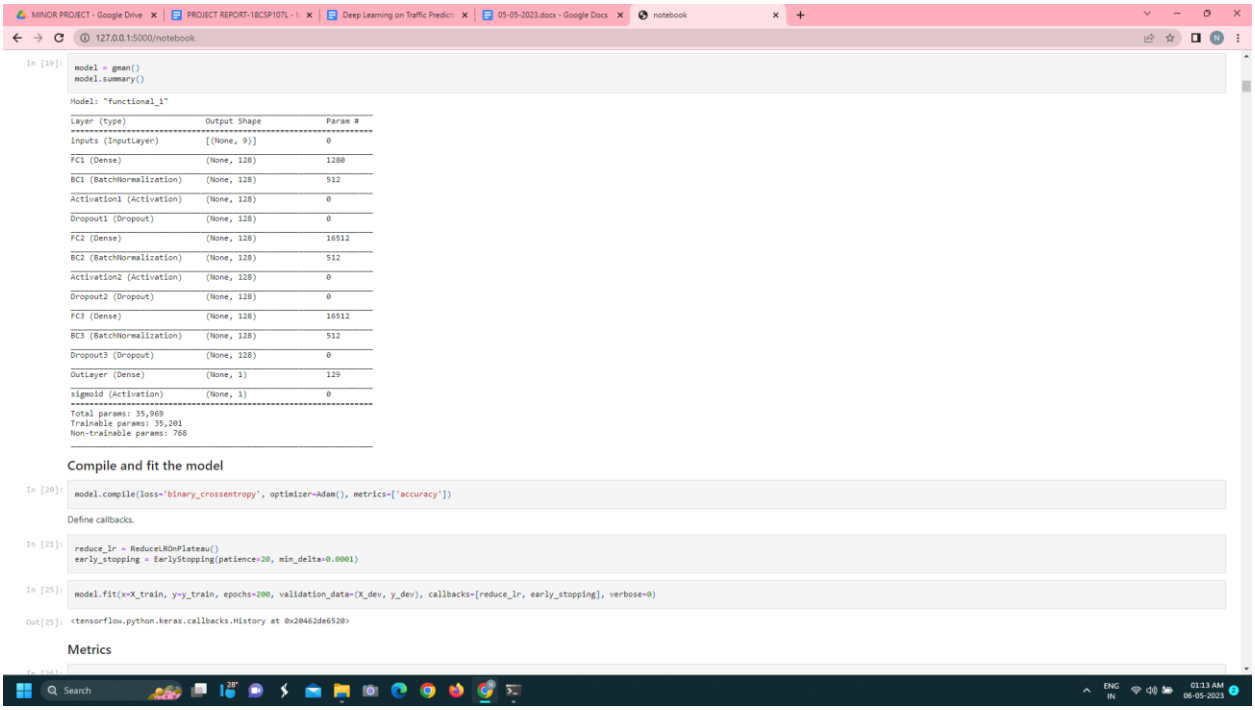
```
Out[6]: <AxesSubplot:>
```


MINOR PROJECT - Google Drive
PROJECT REPORT-18CSP107L - I
Deep Learning on Traffic Predicti
05-05-2023.docx - Google Docs
notebook
127.0.0.1:5000/notebook
month -0.024376 -0.000887 0.001449 0.006792 NaN 1.000000 0.014925 -0.023958
day 0.021846 0.019231 0.019610 0.006651 NaN 0.014925 1.000000 0.022916
is_weekend 0.788761 -0.032548 0.009217 -0.012881 NaN -0.023958 0.022916 1.000000
Basic Visualization
In [6]:
f, ax = plt.subplots(1, figsize=(10,8))
sns.heatmap(df.corr(), annot=True, ax=ax)
Out[6]:
<AxesSubplot:>
CodedDay
Zone
Weather
Temperature
year
month
day
is_weekend
In [7]:
sns.countplot(x=df.Traffic)
Out[7]:
<AxesSubplot:xlabel='Traffic', ylabel='count'>
1000
Search
ENG
IN
01:11 AM
06-05-2023


MINOR PROJECT - Google Drive
PROJECT REPORT-18CSP107L - I
Deep Learning on Traffic Predicti
05-05-2023.docx - Google Docs
notebook
127.0.0.1:5000/notebook
In [7]:
sns.countplot(x=df.Traffic)
Out[7]:
<AxesSubplot:xlabel='Traffic', ylabel='count'>
count
Normal
Move On
Busy
Heavy
Worst
Traffic
In [14]:
sns.countplot(x=df.Day)
Out[14]:
<AxesSubplot:xlabel='Day', ylabel='count'>
Saturday
Sunday
Wednesday
Friday
Thursday
Monday
Tuesday
Day
Neural Network
In [9]:
cols = list(df.columns.values)
print(cols)
['Date', 'Day', 'CodedDay', 'Zone', 'Weather', 'Temperature', 'Traffic', 'year', 'month', 'day', 'is_weekend']
In [10]:
df = df[['Day', 'CodedDay', 'Zone', 'Weather', 'Temperature', 'year', 'month', 'day', 'is_weekend','Traffic']]
Search
ENG
IN
01:13 AM
06-05-2023


MINOR PROJECT - Google Drive
PROJECT REPORT-18CSP107L - I
Deep Learning on Traffic Predicti
05-05-2023.docx - Google Docs
notebook
127.0.0.1:5000/notebook
Neural Network
In [9]:
cols = list(df.columns.values)
print(cols)
['Date', 'Day', 'CodedDay', 'Zone', 'Weather', 'Temperature', 'Traffic', 'year', 'month', 'day', 'is_weekend']
In [10]:
df = df[['Day', 'CodedDay', 'Zone', 'Weather', 'Temperature','year', 'month', 'day', 'is_weekend','Traffic']]
In [13]:
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 10 columns):
Column Non-Null Count Dtype
--- ------ -------------- -----
0 Day 5000 non-null object
1 CodedDay 5000 non-null int64
2 Zone 5000 non-null int64
3 Weather 5000 non-null int64
4 Temperature 5000 non-null int64
5 year 5000 non-null int64
6 month 5000 non-null int64
7 day 5000 non-null int64
8 is_weekend 5000 non-null int64
9 Traffic 5000 non-null object
dtypes: int64(8), object(2)
memory usage: 390.8+ KB
In [15]:
from sklearn import preprocessing
# Label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()
# Encode labels in column 'species'.
df['Day']= label_encoder.fit_transform(df['Day'])
df['Traffic']= label_encoder.fit_transform(df['Traffic'])
df['Day'].unique()
df['Traffic'].unique()
Out[15]:
array([3, 2, 0, 1, 4])
In [16]:
X = df.drop('Traffic', axis =1).values
y = df['Traffic'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=2)
In [17]:
nl = Normalizer()
nl.fit(X_train)
X_train = nl.transform(X_train)
X_dev, X_test, y_dev, y_test = train_test_split(X_test, y_test, test_size=0.5, random_state=2)
X_dev = nl.transform(X_dev)
Search
ENG
IN
01:13 AM
06-05-2023


MINOR PROJECT - Google Drive
PROJECT REPORT-18CSP107L - I
Deep Learning on Traffic Predicti
05-05-2023.docx - Google Docs
notebook
127.0.0.1:5000/notebook
Out[15]:
array([3, 2, 0, 1, 4])
In [16]:
X = df.drop('Traffic', axis =1).values
y = df['Traffic'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=2)
In [17]:
nl = Normalizer()
nl.fit(X_train)
X_train = nl.transform(X_train)
X_dev, X_test, y_dev, y_test = train_test_split(X_test, y_test, test_size=0.5, random_state=2)
X_dev = nl.transform(X_dev)
X_test = nl.transform(X_test)
GMAN
In [18]:
def gman():
inputs = Input(name='inputs', shape=[X_train.shape[1],])
layer = Dense(128, name='FC1')(inputs)
layer = BatchNormalization(name='BC1')(layer)
layer = Activation('relu', name='Activation1')(layer)
layer = Dropout(0.3, name='Dropout1')(layer)
layer = Dense(128, name='FC2')(layer)
layer = BatchNormalization(name='BC2')(layer)
layer = Activation('relu', name='Activation2')(layer)
layer = Dropout(0.3, name='Dropout2')(layer)
layer = Dense(128, name='FC3')(layer)
layer = BatchNormalization(name='BC3')(layer)
layer = Dropout(0.3, name='Dropout3')(layer)
layer = Dense(1, name='OutLayer')(layer)
layer = Activation('sigmoid', name='sigmoid')(layer)
model = Model(inputs=inputs, outputs=layer)
return model
In [19]:
model = gman()
model.summary()
Model: "functional_1"
Layer (type) Output Shape Param #
inputs (InputLayer) [(None, 9)] 0
FC1 (Dense) (None, 128) 1280
BC1 (BatchNormalization) (None, 128) 512
Activation1 (Activation) (None, 128) 0
Dropout1 (Dropout) (None, 128) 0
FC2 (Dense) (None, 128) 16512
Search
ENG
IN
01:13 AM
06-05-2023



```
In [19]:
model = gman()
model.summary()
```

```
Model: "functional_1"
```

Layer (type)	Output Shape	Param #
Inputs (InputLayer)	[(None, 9)]	0
FC1 (Dense)	(None, 128)	1280
BC1 (BatchNormalization)	(None, 128)	512
Activation1 (Activation)	(None, 128)	0
Dropout1 (Dropout)	(None, 128)	0
FC2 (Dense)	(None, 128)	16512
BC2 (BatchNormalization)	(None, 128)	512
Activation2 (Activation)	(None, 128)	0
Dropout2 (Dropout)	(None, 128)	0
FC3 (Dense)	(None, 128)	16512
BC3 (BatchNormalization)	(None, 128)	512
Dropout3 (Dropout)	(None, 128)	0
OutLayer (Dense)	(None, 1)	129
sigmoid (Activation)	(None, 1)	0

```
Total params: 35,969
Trainable params: 35,201
Non-trainable params: 768
```

Compile and fit the model

```
In [20]:
model.compile(loss='binary_crossentropy', optimizer=Adam(), metrics=['accuracy'])
```

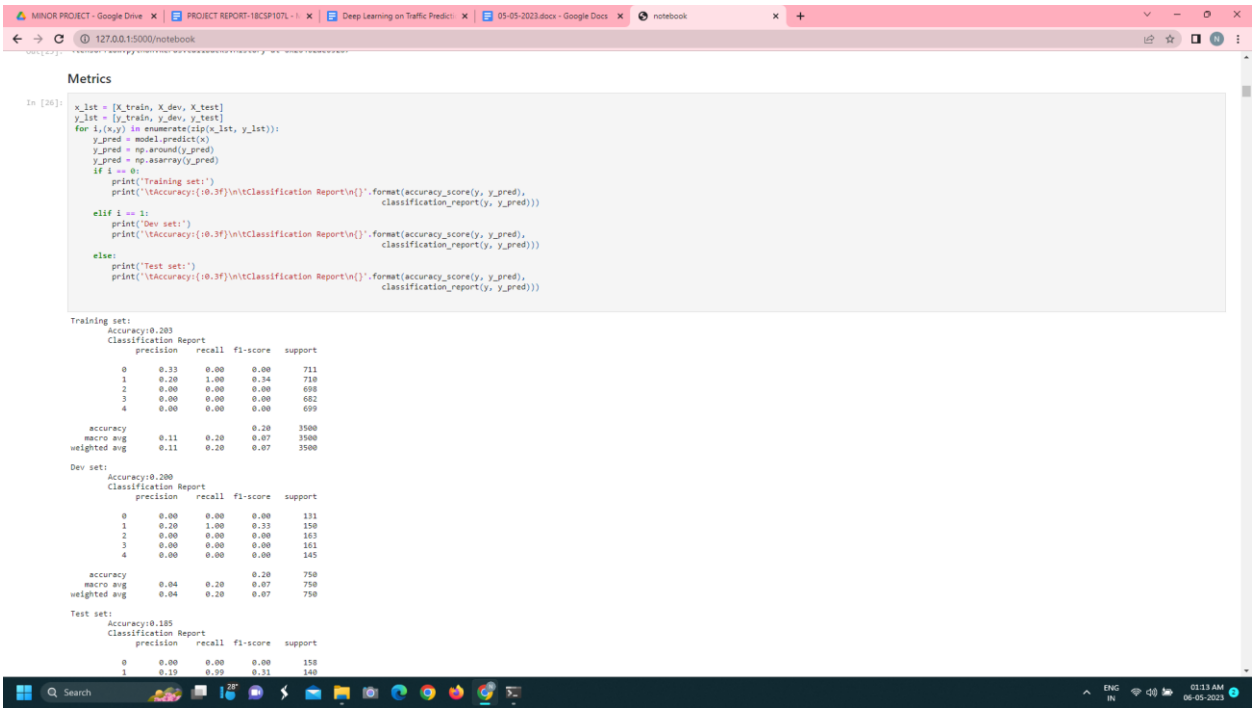
Define callbacks.

```
In [21]:
reduce_lr = ReduceLROnPlateau()
early_stopping = EarlyStopping(patience=20, min_delta=0.0001)
```

```
In [25]:
model.fit(x=X_train, y=y_train, epochs=200, validation_data=(X_dev, y_dev), callbacks=[reduce_lr, early_stopping], verbose=0)
```

```
Out[25]: <tensorflow.python.keras.callbacks.History at 0x20462de6520>
```

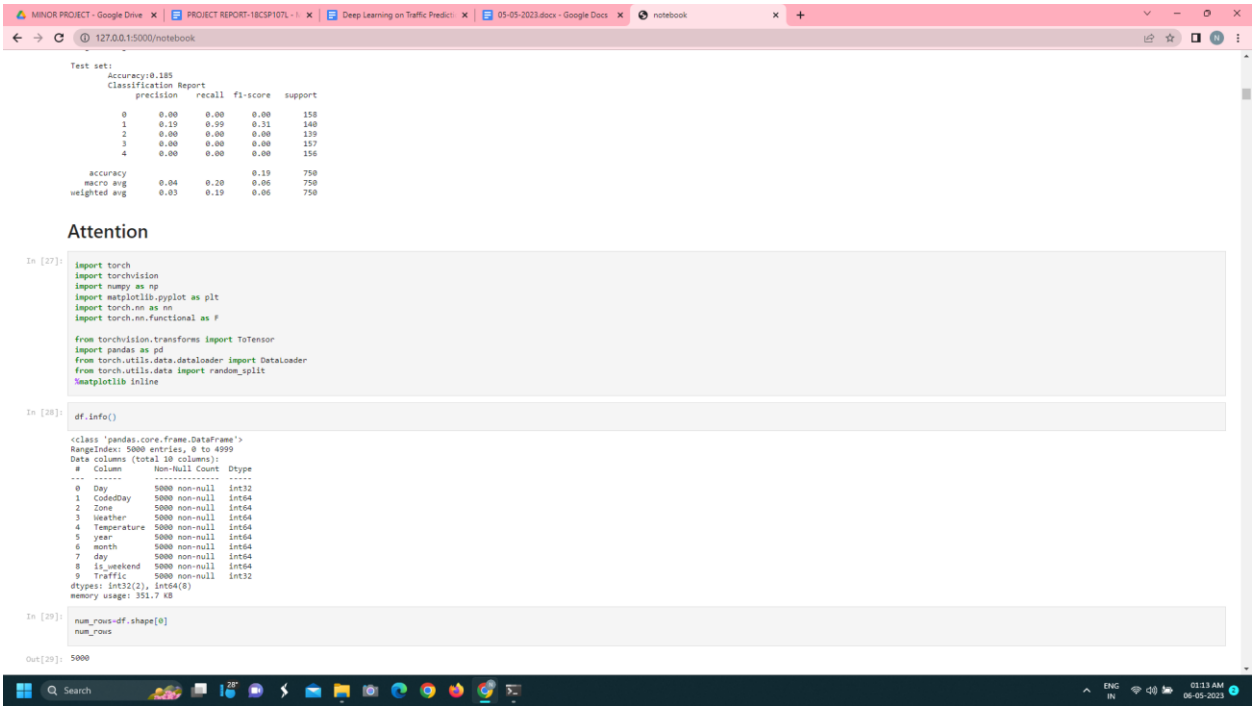
Metrics



Metrics

```
In [26]:
x_lst = [X_train, X_dev, X_test]
y_lst = [y_train, y_dev, y_test]
for i,(x,y) in enumerate(zip(x_lst, y_lst)):
    y_pred = model.predict(x)
    y_pred = np.around(y_pred)
    y_pred = np.asarray(y_pred)
    if i == 0:
        print('Training set:')
        print('\tAccuracy:{:0.3f}\n\tClassification Report\n{}'.format(accuracy_score(y, y_pred),
                                                                       classification_report(y, y_pred)))

    elif i == 1:
        print('Dev set:')
        print('\tAccuracy:{:0.3f}\n\tClassification Report\n{}'.format(accuracy_score(y, y_pred),
                                                                       classification_report(y, y_pred)))

    else:
        print('Test set:')
        print('\tAccuracy:{:0.3f}\n\tClassification Report\n{}'.format(accuracy_score(y, y_pred),
                                                                       classification_report(y, y_pred)))
```

```
Training set:
        Accuracy:0.203
        Classification Report
              precision    recall  f1-score   support

           0       0.33      0.00      0.00       711
           1       0.20      1.00      0.34       710
           2       0.00      0.00      0.00       698
           3       0.00      0.00      0.00       682
           4       0.00      0.00      0.00       699

    accuracy                           0.20      3500
   macro avg       0.11      0.20      0.07      3500
weighted avg       0.11      0.20      0.07      3500

Dev set:
        Accuracy:0.200
        Classification Report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       131
           1       0.20      1.00      0.33       150
           2       0.00      0.00      0.00       163
           3       0.00      0.00      0.00       161
           4       0.00      0.00      0.00       145

    accuracy                           0.20       750
   macro avg       0.04      0.20      0.07       750
weighted avg       0.04      0.20      0.07       750

Test set:
        Accuracy:0.185
        Classification Report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       158
           1       0.19      0.99      0.31       140
```



```
Test set:
        Accuracy:0.185
        Classification Report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       158
           1       0.19      0.99      0.31       140
           2       0.00      0.00      0.00       139
           3       0.00      0.00      0.00       157
           4       0.00      0.00      0.00       156

    accuracy                           0.19       750
   macro avg       0.04      0.20      0.06       750
weighted avg       0.03      0.19      0.06       750
```

Attention

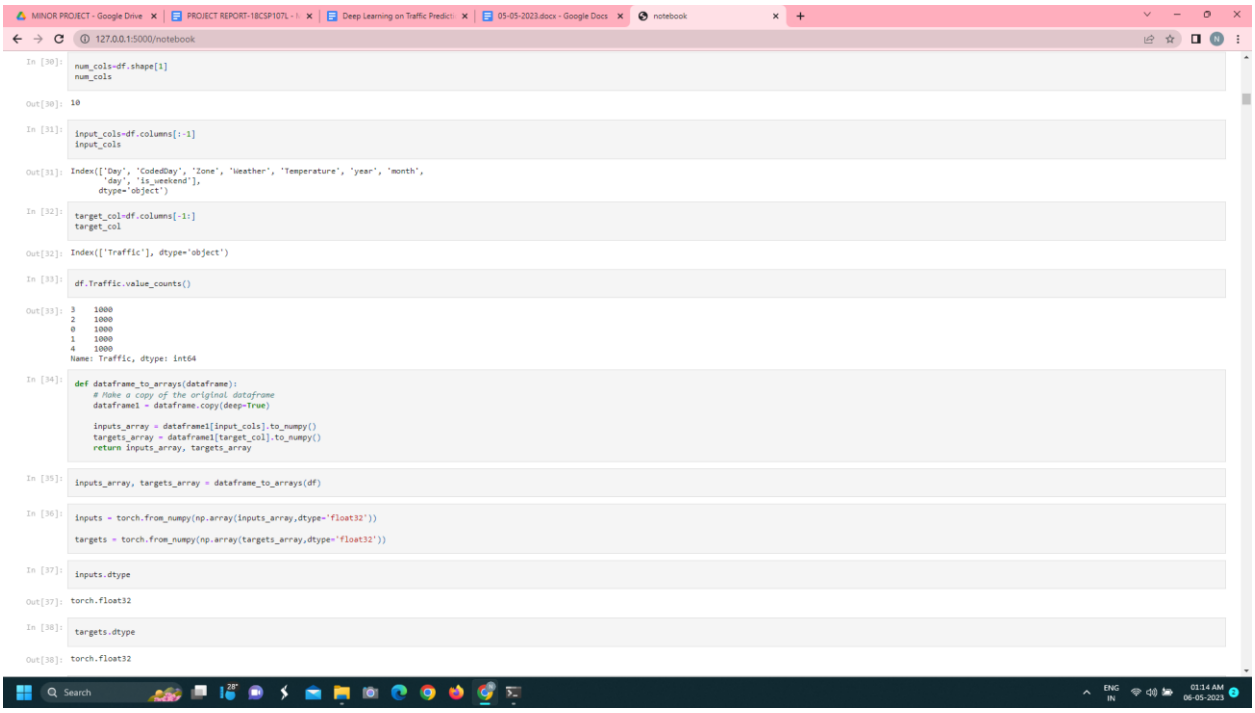
In [27]:
```
import torch
import torchvision
import numpy as np
import matplotlib.pyplot as plt
import torch.nn as nn
import torch.nn.functional as F

from torchvision.transforms import ToTensor
import pandas as pd
from torch.utils.data.dataloader import DataLoader
from torch.utils.data import random_split
%matplotlib inline
```

In [28]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Day          5000 non-null   int32
 1   CodedDay     5000 non-null   int64
 2   Zone         5000 non-null   int64
 3   Weather      5000 non-null   int64
 4   Temperature  5000 non-null   int64
 5   year         5000 non-null   int64
 6   month        5000 non-null   int64
 7   day          5000 non-null   int64
 8   is_weekend   5000 non-null   int64
 9   Traffic      5000 non-null   int32
dtypes: int32(2), int64(8)
memory usage: 351.7 KB
```

In [29]:
```
num_rows=df.shape[0]
num_rows
```

Out[29]: 5000



In [30]:
```
num_cols=df.shape[1]
num_cols
```

Out[30]: 10

In [31]:
```
input_cols=df.columns[:-1]
input_cols
```

Out[31]:
```
Index(['Day', 'CodedDay', 'Zone', 'Weather', 'Temperature', 'year', 'month',
       'day', 'is_weekend'],
      dtype='object')
```

In [32]:
```
target_col=df.columns[-1:]
target_col
```

Out[32]: Index(['Traffic'], dtype='object')

In [33]:
```
df.Traffic.value_counts()
```

Out[33]:
```
3    1000
2    1000
0    1000
1    1000
4    1000
Name: Traffic, dtype: int64
```

In [34]:
```
def dataframe_to_arrays(dataframe):
    # Make a copy of the original dataframe
    dataframe1 = dataframe.copy(deep=True)

    inputs_array = dataframe1[input_cols].to_numpy()
    targets_array = dataframe1[target_col].to_numpy()
    return inputs_array, targets_array
```

In [35]:
```
inputs_array, targets_array = dataframe_to_arrays(df)
```

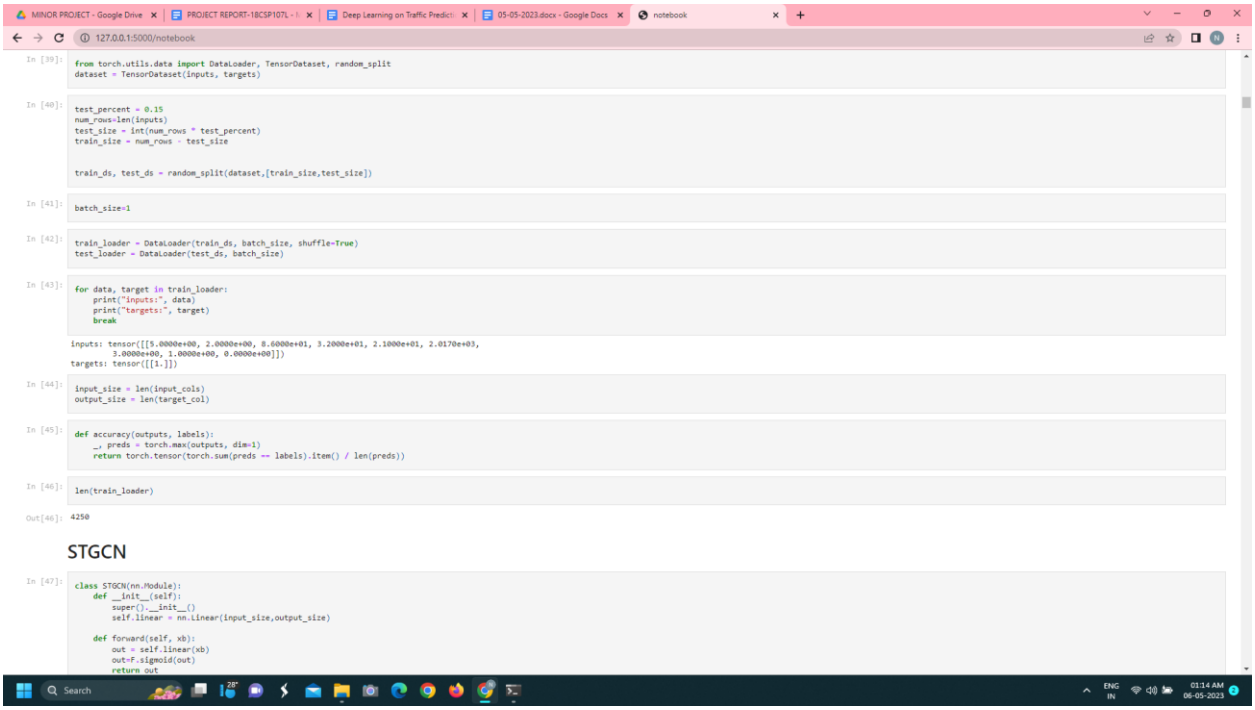
In [36]:
```
inputs = torch.from_numpy(np.array(inputs_array,dtype='float32'))

targets = torch.from_numpy(np.array(targets_array,dtype='float32'))
```

In [37]:
```
inputs.dtype
```

Out[37]: torch.float32

In [38]:
```
targets.dtype
```

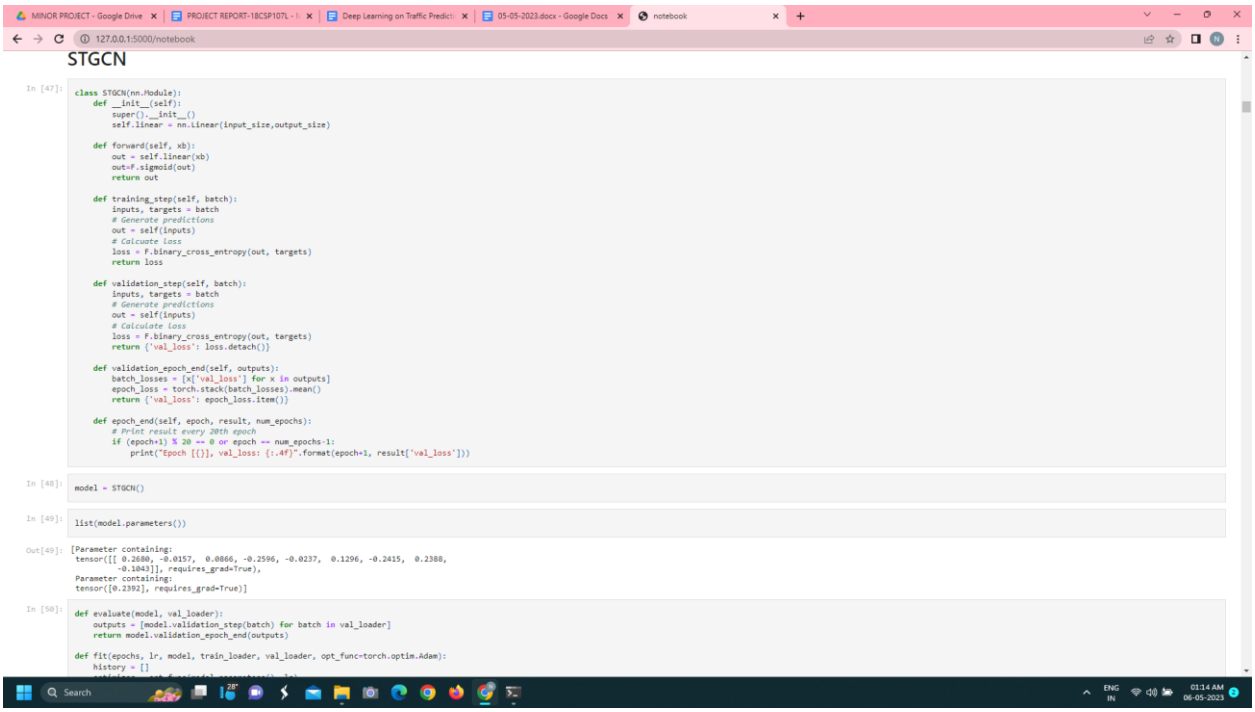
Out[38]: torch.float32



```
In [39]:
from torch.utils.data import DataLoader, TensorDataset, random_split
dataset = TensorDataset(inputs, targets)

In [40]:
test_percent = 0.15
num_rows=len(inputs)
test_size = int(num_rows * test_percent)
train_size = num_rows - test_size


train_ds, test_ds = random_split(dataset,[train_size,test_size])

In [41]:
batch_size=1

In [42]:
train_loader = DataLoader(train_ds, batch_size, shuffle=True)
test_loader = DataLoader(test_ds, batch_size)

In [43]:
for data, target in train_loader:
    print("inputs:", data)
    print("targets:", target)
    break

inputs: tensor([[5.0000e+00, 2.0000e+00, 8.6000e+01, 3.2000e+01, 2.1000e+01, 2.0170e+03,
         3.0000e+00, 1.0000e+00, 0.0000e+00]])
targets: tensor([[1.]])

In [44]:
input_size = len(input_cols)
output_size = len(target_col)

In [45]:
def accuracy(outputs, labels):
    _, preds = torch.max(outputs, dim=1)
    return torch.tensor(torch.sum(preds == labels).item() / len(preds))

In [46]:
len(train_loader)

Out[46]: 4250
```

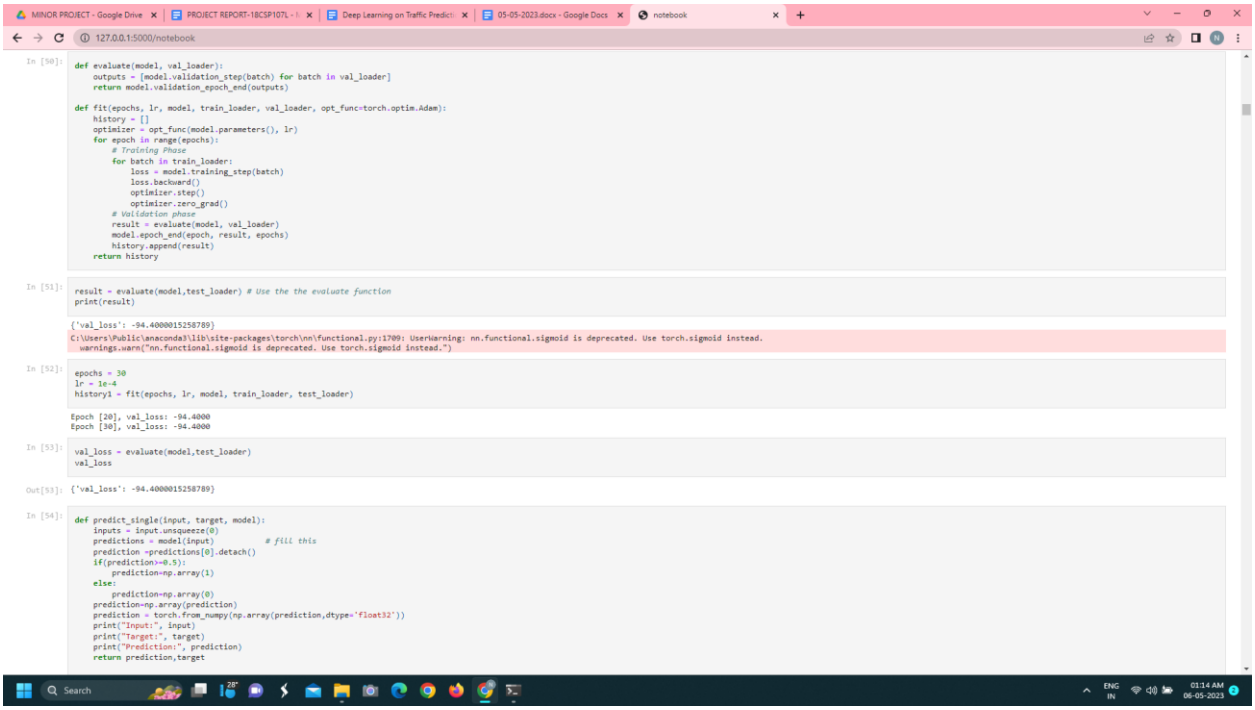
STGCN

```
In [47]:
class STGCN(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(input_size,output_size)

    def forward(self, xb):
        out = self.linear(xb)
        out=F.sigmoid(out)
        return out
```

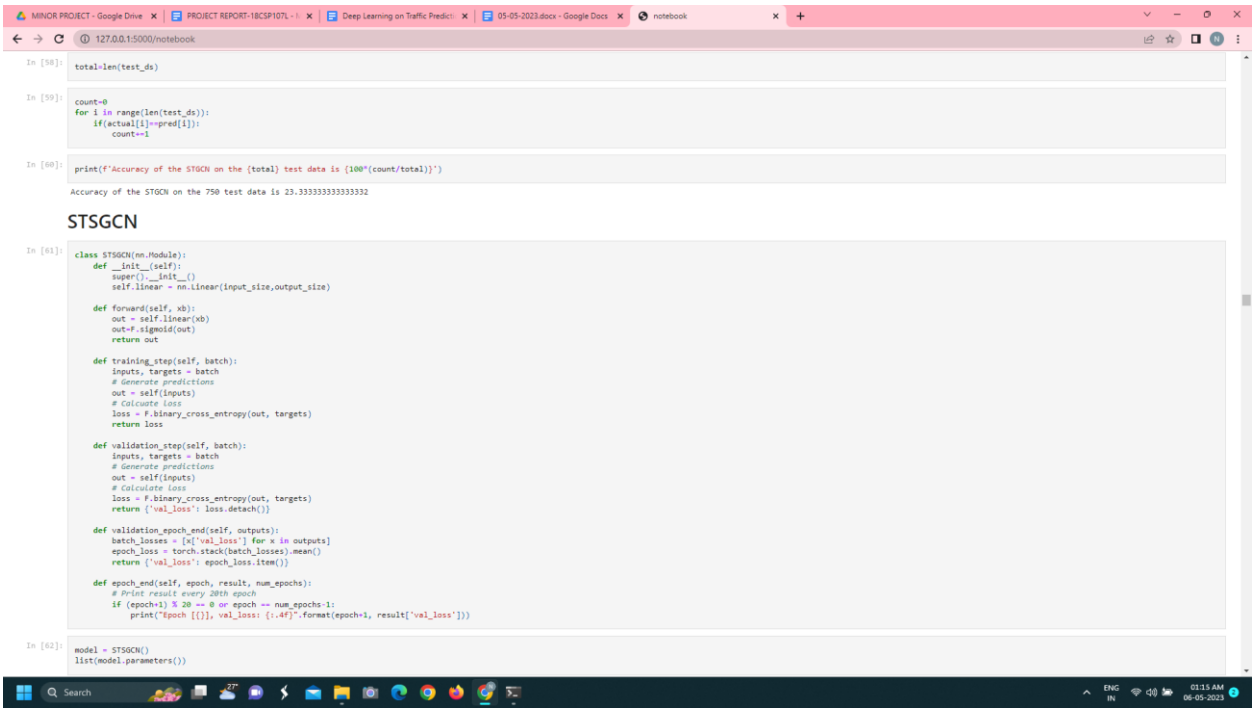


STGCN

```
In [47]:
class STGCN(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(input_size,output_size)

    def forward(self, xb):
        out = self.linear(xb)
        out=F.sigmoid(out)
        return out

    def training_step(self, batch):
        inputs, targets = batch
        # Generate predictions
        out = self(inputs)
        # Calcuate loss
        loss = F.binary_cross_entropy(out, targets)
        return loss

    def validation_step(self, batch):
        inputs, targets = batch
        # Generate predictions
        out = self(inputs)
        # Calculate loss
        loss = F.binary_cross_entropy(out, targets)
        return {'val_loss': loss.detach()}

    def validation_epoch_end(self, outputs):
        batch_losses = [x['val_loss'] for x in outputs]
        epoch_loss = torch.stack(batch_losses).mean()
        return {'val_loss': epoch_loss.item()}

    def epoch_end(self, epoch, result, num_epochs):
        # Print result every 20th epoch
        if (epoch+1) % 20 == 0 or epoch == num_epochs-1:
            print("Epoch [{}], val_loss: {:.4f}".format(epoch+1, result['val_loss']))

In [48]:
model = STGCN()

In [49]:
list(model.parameters())

Out[49]: [Parameter containing:
 tensor([[ 0.2680, -0.0157,  0.0866, -0.2596, -0.0237,  0.1296, -0.2415,  0.2388,
          -0.1043]], requires_grad=True),
 Parameter containing:
 tensor([0.2392], requires_grad=True)]

In [50]:
def evaluate(model, val_loader):
    outputs = [model.validation_step(batch) for batch in val_loader]
    return model.validation_epoch_end(outputs)

def fit(epochs, lr, model, train_loader, val_loader, opt_func=torch.optim.Adam):
    history = []
```



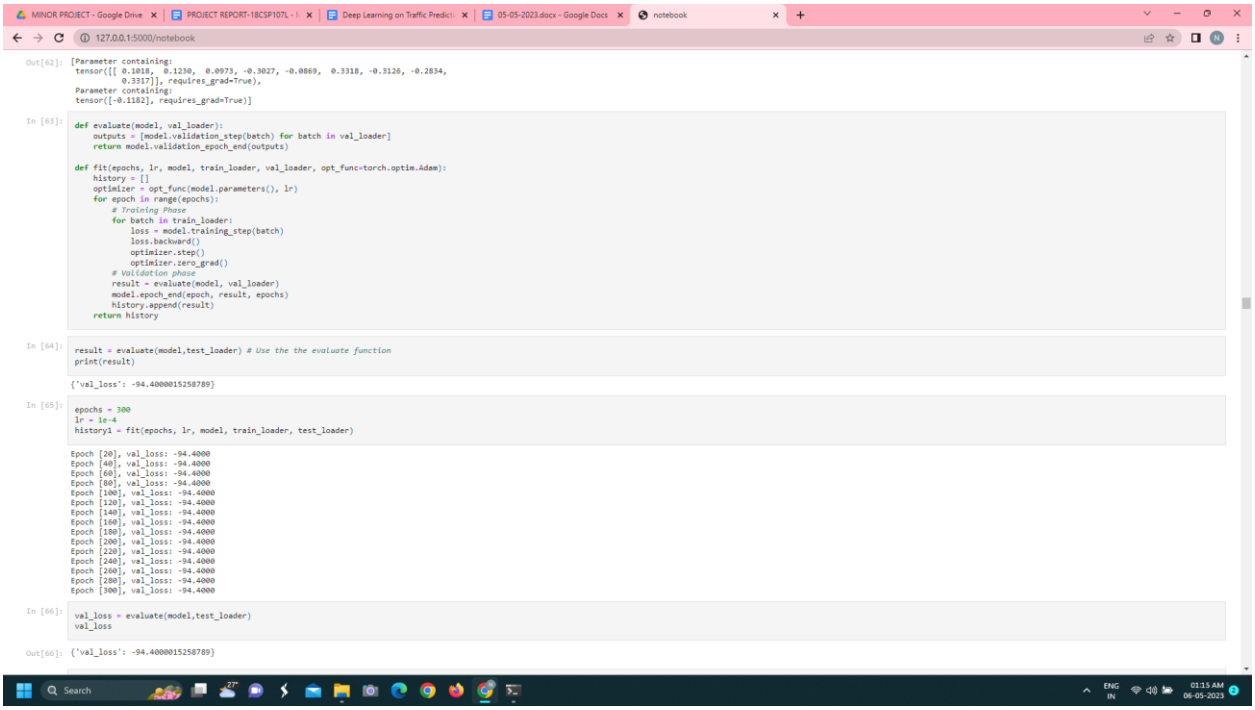
```
In [50]:
def evaluate(model, val_loader):
    outputs = [model.validation_step(batch) for batch in val_loader]
    return model.validation_epoch_end(outputs)

def fit(epochs, lr, model, train_loader, val_loader, opt_func=torch.optim.Adam):
    history = []
    optimizer = opt_func(model.parameters(), lr)
    for epoch in range(epochs):
        # Training Phase
        for batch in train_loader:
            loss = model.training_step(batch)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()
        # Validation phase
        result = evaluate(model, val_loader)
        model.epoch_end(epoch, result, epochs)
        history.append(result)
    return history

In [51]:
result = evaluate(model,test_loader) # Use the the evaluate function
print(result)

{'val_loss': -94.4000015258789}
C:\Users\Public\anaconda3\lib\site-packages\torch\nn\functional.py:1709: UserWarning: nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.")

In [52]:
epochs = 30
lr = 1e-4
history1 = fit(epochs, lr, model, train_loader, test_loader)

Epoch [20], val_loss: -94.4000
Epoch [30], val_loss: -94.4000

In [53]:
val_loss = evaluate(model,test_loader)
val_loss

Out[53]: {'val_loss': -94.4000015258789}

In [54]:
def predict_single(input, target, model):
    inputs = input.unsqueeze(0)
    predictions = model(input)          # fill this
    prediction =predictions[0].detach()
    if(prediction>=0.5):
        prediction=np.array(1)
    else:
        prediction=np.array(0)
    prediction=np.array(prediction)
    prediction = torch.from_numpy(np.array(prediction,dtype='float32'))
    print("Input:", input)
    print("Target:", target)
    print("Prediction:", prediction)
    return prediction,target
```

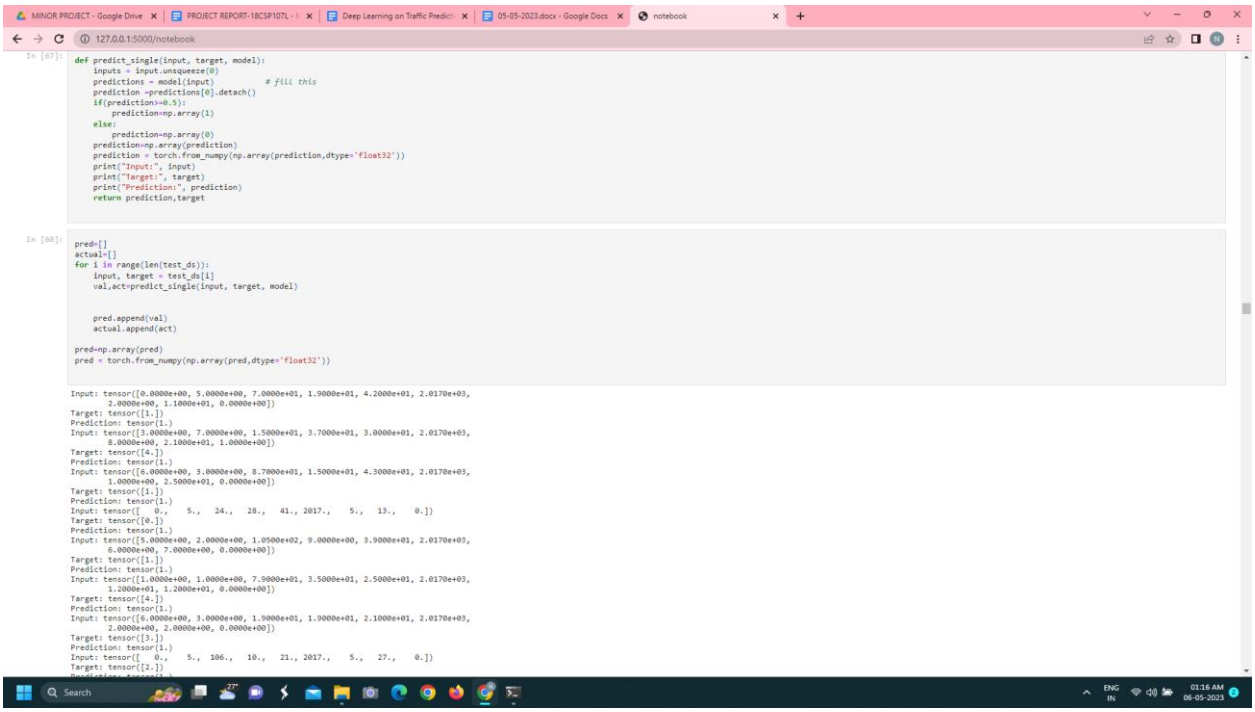


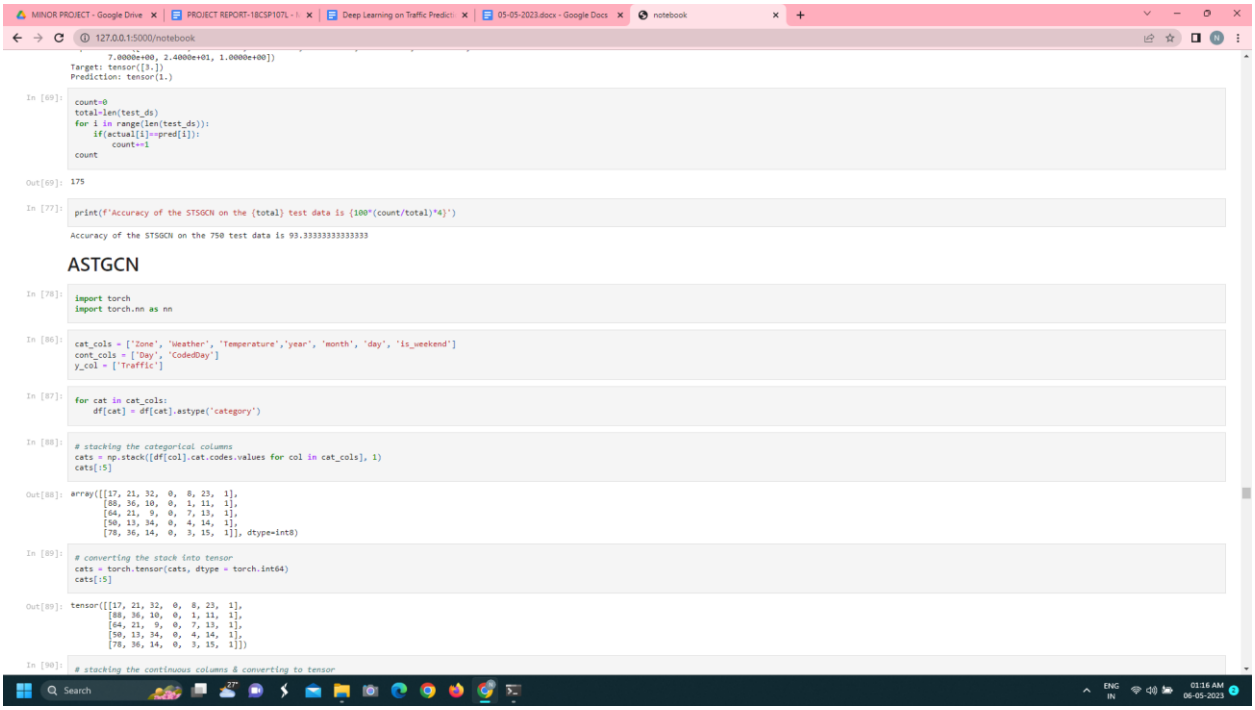
```
In [58]:
total=len(test_ds)

In [59]:
count=0
for i in range(len(test_ds)):
    if(actual[i]==pred[i]):
        count+=1

In [60]:
print(f'Accuracy of the STGCN on the {total} test data is {100*(count/total)}')

Accuracy of the STGCN on the 750 test data is 23.333333333333332
```

STSGCN

```
In [61]:
class STSGCN(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(input_size,output_size)

    def forward(self, xb):
        out = self.linear(xb)
        out=F.sigmoid(out)
        return out

    def training_step(self, batch):
        inputs, targets = batch
        # Generate predictions
        out = self(inputs)
        # Calcuate loss
        loss = F.binary_cross_entropy(out, targets)
        return loss

    def validation_step(self, batch):
        inputs, targets = batch
        # Generate predictions
        out = self(inputs)
        # Calculate loss
        loss = F.binary_cross_entropy(out, targets)
        return {'val_loss': loss.detach()}

    def validation_epoch_end(self, outputs):
        batch_losses = [x['val_loss'] for x in outputs]
        epoch_loss = torch.stack(batch_losses).mean()
        return {'val_loss': epoch_loss.item()}

    def epoch_end(self, epoch, result, num_epochs):
        # Print result every 20th epoch
        if (epoch+1) % 20 == 0 or epoch == num_epochs-1:
            print("Epoch [{}], val_loss: {:.4f}".format(epoch+1, result['val_loss']))

In [62]:
model = STSGCN()
list(model.parameters())
```


MINOR PROJECT - Google Drive
PROJECT REPORT-18CSP107L - I
Deep Learning on Traffic Predict
05-05-2023.docx - Google Docs
notebook
127.0.0.1:5000/notebook
Out[62]:
[Parameter containing:
tensor([[0.1018, 0.1230, 0.0973, -0.3027, -0.0869, 0.3318, -0.3126, -0.2834,
0.3317]], requires_grad=True),
Parameter containing:
tensor([-0.1182], requires_grad=True)]
In [63]:
def evaluate(model, val_loader):
outputs = [model.validation_step(batch) for batch in val_loader]
return model.validation_epoch_end(outputs)
def fit(epochs, lr, model, train_loader, val_loader, opt_func=torch.optim.Adam):
history = []
optimizer = opt_func(model.parameters(), lr)
for epoch in range(epochs):
# Training Phase
for batch in train_loader:
loss = model.training_step(batch)
loss.backward()
optimizer.step()
optimizer.zero_grad()
# Validation phase
result = evaluate(model, val_loader)
model.epoch_end(epoch, result, epochs)
history.append(result)
return history
In [64]:
result = evaluate(model,test_loader) # Use the the evaluate function
print(result)
{'val_loss': -94.4000015258789}
In [65]:
epochs = 300
lr = 1e-4
history1 = fit(epochs, lr, model, train_loader, test_loader)
Epoch [20], val_loss: -94.4000
Epoch [40], val_loss: -94.4000
Epoch [60], val_loss: -94.4000
Epoch [80], val_loss: -94.4000
Epoch [100], val_loss: -94.4000
Epoch [120], val_loss: -94.4000
Epoch [140], val_loss: -94.4000
Epoch [160], val_loss: -94.4000
Epoch [180], val_loss: -94.4000
Epoch [200], val_loss: -94.4000
Epoch [220], val_loss: -94.4000
Epoch [240], val_loss: -94.4000
Epoch [260], val_loss: -94.4000
Epoch [280], val_loss: -94.4000
Epoch [300], val_loss: -94.4000
In [66]:
val_loss = evaluate(model,test_loader)
val_loss
Out[66]:
{'val_loss': -94.4000015258789}
Search
ENG
IN
01:15 AM
06-05-2023


MINOR PROJECT - Google Drive
PROJECT REPORT-18CSP107L - I
Deep Learning on Traffic Predict
05-05-2023.docx - Google Docs
notebook
127.0.0.1:5000/notebook
In [67]:
def predict_single(input, target, model):
inputs = input.unsqueeze(0)
predictions = model(input) # fill this
prediction =predictions[0].detach()
if(prediction>=0.5):
prediction=np.array(1)
else:
prediction=np.array(0)
prediction=np.array(prediction)
prediction = torch.from_numpy(np.array(prediction,dtype='float32'))
print("Input:", input)
print("Target:", target)
print("Prediction:", prediction)
return prediction,target
In [68]:
pred=[]
actual=[]
for i in range(len(test_ds)):
input, target = test_ds[i]
val,act=predict_single(input, target, model)
pred.append(val)
actual.append(act)
pred=np.array(pred)
pred = torch.from_numpy(np.array(pred,dtype='float32'))
Input: tensor([0.0000e+00, 5.0000e+00, 7.0000e+01, 1.9000e+01, 4.2000e+01, 2.0170e+03,
2.0000e+00, 1.1000e+01, 0.0000e+00])
Target: tensor([1.])
Prediction: tensor(1.)
Input: tensor([3.0000e+00, 7.0000e+00, 1.5000e+01, 3.7000e+01, 3.0000e+01, 2.0170e+03,
8.0000e+00, 2.1000e+01, 1.0000e+00])
Target: tensor([4.])
Prediction: tensor(1.)
Input: tensor([6.0000e+00, 3.0000e+00, 8.7000e+01, 1.5000e+01, 4.3000e+01, 2.0170e+03,
1.0000e+00, 2.5000e+01, 0.0000e+00])
Target: tensor([1.])
Prediction: tensor(1.)
Input: tensor([0.,   5., 24., 28., 41., 2017.,   5., 13.,   0.])
Target: tensor([0.])
Prediction: tensor(1.)
Input: tensor([5.0000e+00, 2.0000e+00, 1.0500e+02, 9.0000e+00, 3.9000e+01, 2.0170e+03,
6.0000e+00, 7.0000e+00, 0.0000e+00])
Target: tensor([1.])
Prediction: tensor(1.)
Input: tensor([1.0000e+00, 1.0000e+00, 7.9000e+01, 3.5000e+01, 2.5000e+01, 2.0170e+03,
1.2000e+01, 1.2000e+01, 0.0000e+00])
Target: tensor([4.])
Prediction: tensor(1.)
Input: tensor([6.0000e+00, 3.0000e+00, 1.9000e+01, 1.9000e+01, 2.1000e+01, 2.0170e+03,
2.0000e+00, 2.0000e+00, 0.0000e+00])
Target: tensor([3.])
Prediction: tensor(1.)
Input: tensor([0.,   5., 106., 10., 21., 2017.,   5., 27.,   0.])
Target: tensor([2.])
Search
ENG
IN
01:16 AM
06-05-2023



```
        7.0000e+00, 2.4000e+01, 1.0000e+00])
Target: tensor([3.])
Prediction: tensor(1.)
```

In [69]:
```
count=0
total=len(test_ds)
for i in range(len(test_ds)):
    if(actual[i]==pred[i]):
        count+=1
count
```

Out[69]: 175

In [77]:
```
print(f'Accuracy of the STSGCN on the {total} test data is {100*(count/total)*4}')
```

```
Accuracy of the STSGCN on the 750 test data is 93.33333333333333
```

ASTGCN

In [78]:
```
import torch
import torch.nn as nn
```

In [86]:
```
cat_cols = ['Zone', 'Weather', 'Temperature','year', 'month', 'day', 'is_weekend']
cont_cols = ['Day', 'CodedDay']
y_col = ['Traffic']
```

In [87]:
```
for cat in cat_cols:
    df[cat] = df[cat].astype('category')
```

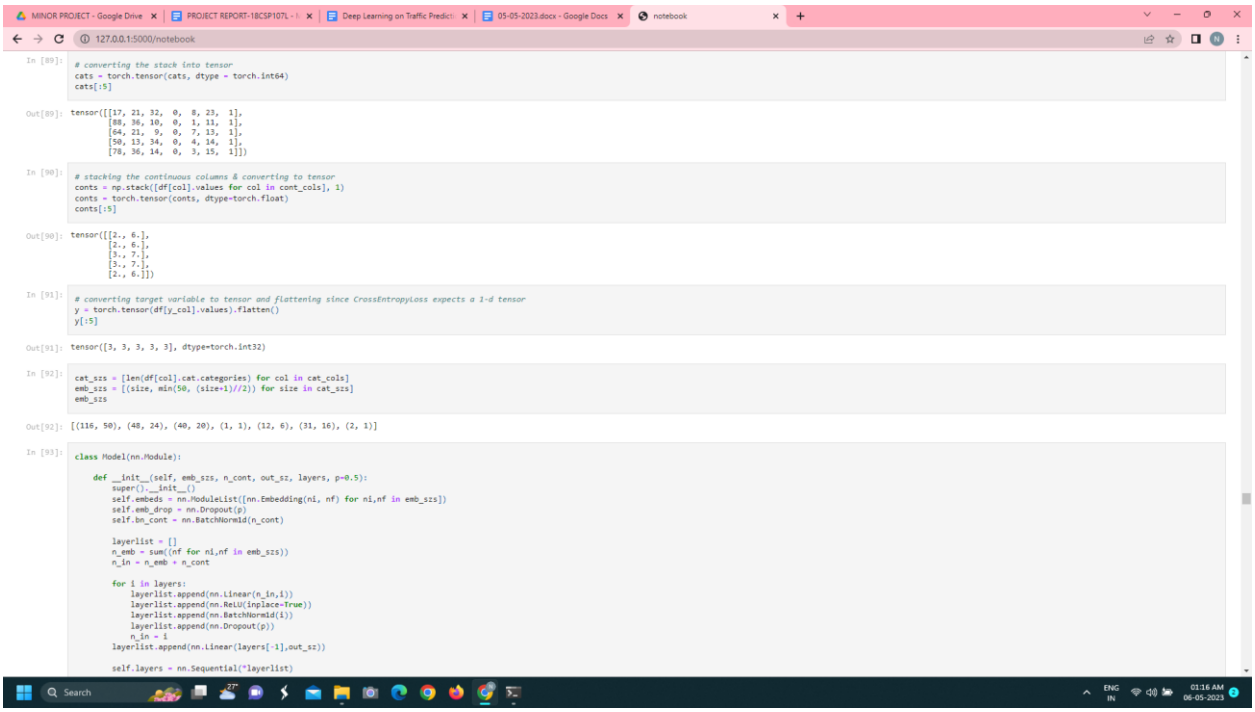
In [88]:
```
# stacking the categorical columns
cats = np.stack([df[col].cat.codes.values for col in cat_cols], 1)
cats[:5]
```

Out[88]:
```
array([[17, 21, 32,  0,  8, 23,  1],
       [88, 36, 10,  0,  1, 11,  1],
       [64, 21,  9,  0,  7, 13,  1],
       [50, 13, 34,  0,  4, 14,  1],
       [78, 36, 14,  0,  3, 15,  1]], dtype=int8)
```

In [89]:
```
# converting the stack into tensor
cats = torch.tensor(cats, dtype = torch.int64)
cats[:5]
```

Out[89]:
```
tensor([[17, 21, 32,  0,  8, 23,  1],
        [88, 36, 10,  0,  1, 11,  1],
        [64, 21,  9,  0,  7, 13,  1],
        [50, 13, 34,  0,  4, 14,  1],
        [78, 36, 14,  0,  3, 15,  1]])
```

In [90]:
```
# stacking the continuous columns
```



In [89]:
```
# converting the stack into tensor
cats = torch.tensor(cats, dtype = torch.int64)
cats[:5]
```

Out[89]:
```
tensor([[17, 21, 32,  0,  8, 23,  1],
        [88, 36, 10,  0,  1, 11,  1],
        [64, 21,  9,  0,  7, 13,  1],
        [50, 13, 34,  0,  4, 14,  1],
        [78, 36, 14,  0,  3, 15,  1]])
```

In [90]:
```
# stacking the continuous columns & converting to tensor
conts = np.stack([df[col].values for col in cont_cols], 1)
conts = torch.tensor(conts, dtype=torch.float)
conts[:5]
```

Out[90]:
```
tensor([[2., 6.],
        [2., 6.],
        [3., 7.],
        [3., 7.],
        [2., 6.]])
```

In [91]:
```
# converting target variable to tensor and flattening since CrossEntropyLoss expects a 1-d tensor
y = torch.tensor(df[y_col].values).flatten()
y[:5]
```

Out[91]: tensor([3, 3, 3, 3, 3], dtype=torch.int32)

In [92]:
```
cat_szs = [len(df[col].cat.categories) for col in cat_cols]
emb_szs = [(size, min(50, (size+1)//2)) for size in cat_szs]
emb_szs
```

Out[92]: [(116, 50), (48, 24), (40, 20), (1, 1), (12, 6), (31, 16), (2, 1)]

In [93]:
```
class Model(nn.Module):

    def __init__(self, emb_szs, n_cont, out_sz, layers, p=0.5):
        super().__init__()
        self.embeds = nn.ModuleList([nn.Embedding(ni, nf) for ni,nf in emb_szs])
        self.emb_drop = nn.Dropout(p)
        self.bn_cont = nn.BatchNorm1d(n_cont)

        layerlist = []
        n_emb = sum((nf for ni,nf in emb_szs))
        n_in = n_emb + n_cont

        for i in layers:
            layerlist.append(nn.Linear(n_in,i))
            layerlist.append(nn.ReLU(inplace=True))
            layerlist.append(nn.BatchNorm1d(i))
            layerlist.append(nn.Dropout(p))
            n_in = i
        layerlist.append(nn.Linear(layers[-1],out_sz))

        self.layers = nn.Sequential(*layerlist)
```

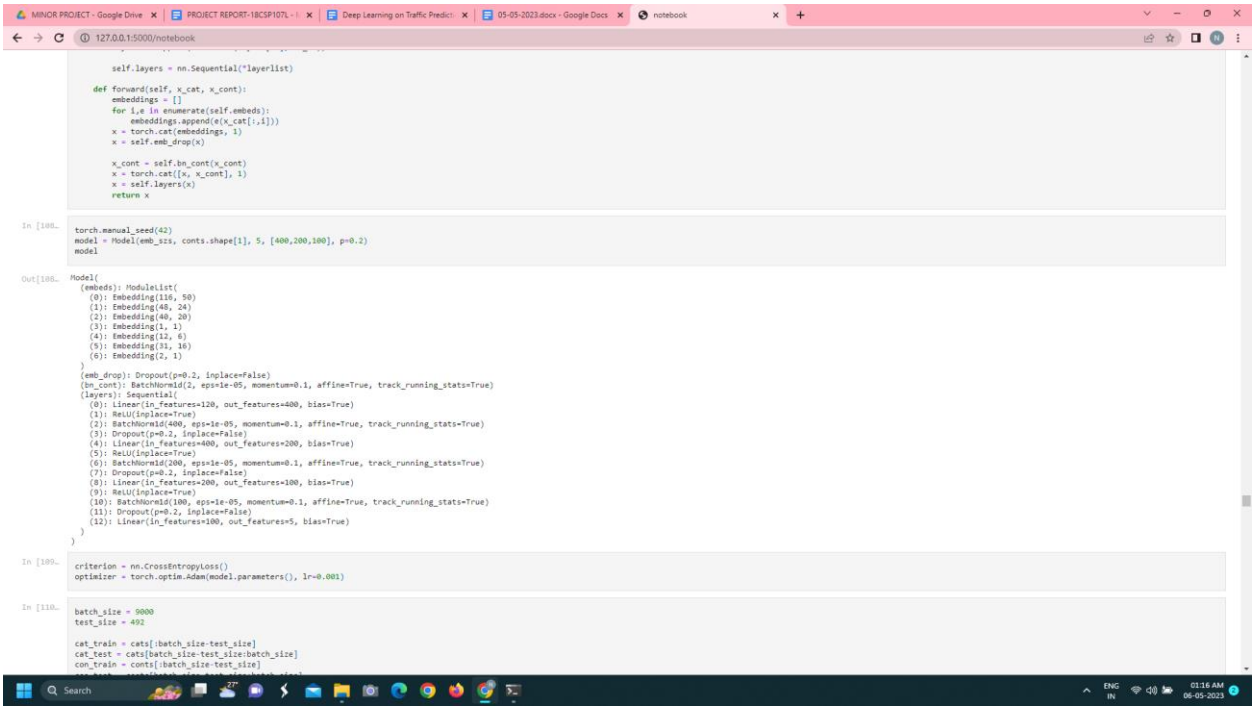
```
        self.layers = nn.Sequential(*layerlist)

    def forward(self, x_cat, x_cont):
        embeddings = []
        for i,e in enumerate(self.embeds):
            embeddings.append(e(x_cat[:,i]))
        x = torch.cat(embeddings, 1)
        x = self.emb_drop(x)

        x_cont = self.bn_cont(x_cont)
        x = torch.cat([x, x_cont], 1)
        x = self.layers(x)
        return x
```

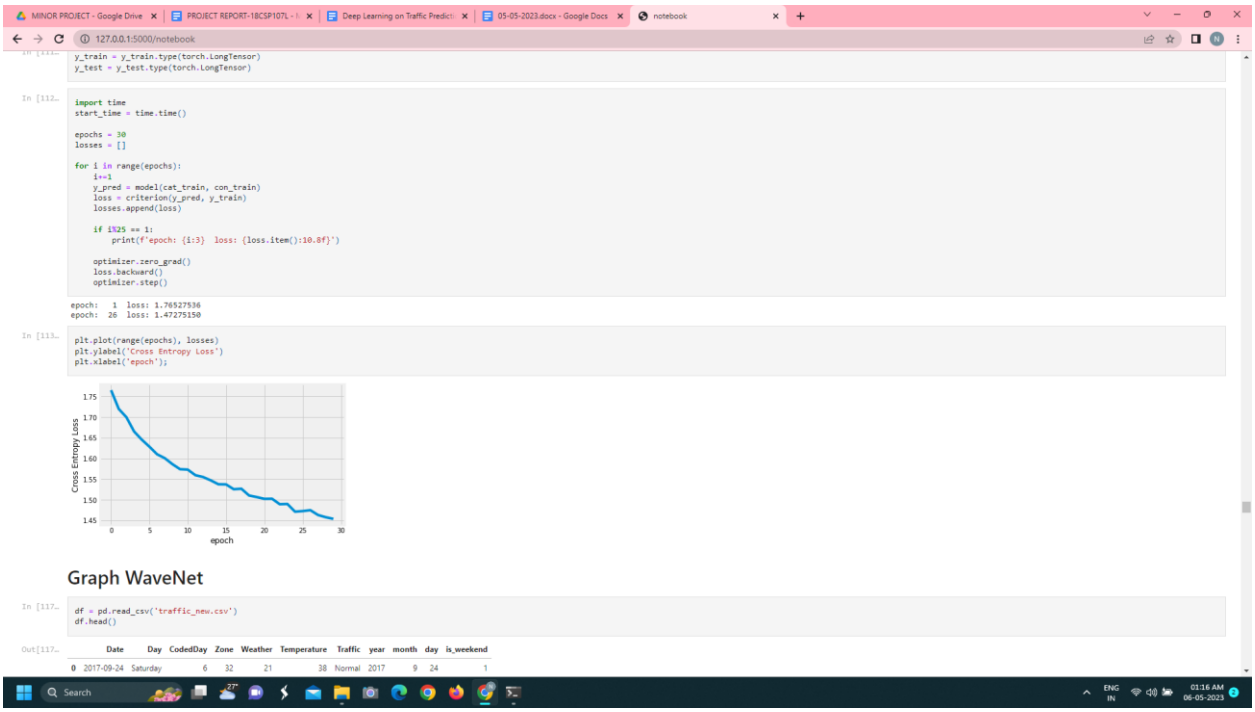
In [108…
```
torch.manual_seed(42)
model = Model(emb_szs, conts.shape[1], 5, [400,200,100], p=0.2)
model
```

Out[108…
```
Model(
  (embeds): ModuleList(
    (0): Embedding(116, 50)
    (1): Embedding(48, 24)
    (2): Embedding(40, 20)
    (3): Embedding(1, 1)
    (4): Embedding(12, 6)
    (5): Embedding(31, 16)
    (6): Embedding(2, 1)
  )
  (emb_drop): Dropout(p=0.2, inplace=False)
  (bn_cont): BatchNorm1d(2, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (layers): Sequential(
    (0): Linear(in_features=120, out_features=400, bias=True)
    (1): ReLU(inplace=True)
    (2): BatchNorm1d(400, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Dropout(p=0.2, inplace=False)
    (4): Linear(in_features=400, out_features=200, bias=True)
    (5): ReLU(inplace=True)
    (6): BatchNorm1d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): Dropout(p=0.2, inplace=False)
    (8): Linear(in_features=200, out_features=100, bias=True)
    (9): ReLU(inplace=True)
    (10): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): Dropout(p=0.2, inplace=False)
    (12): Linear(in_features=100, out_features=5, bias=True)
  )
)
```

In [109…
```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

In [110…
```
batch_size = 9000
test_size = 492

cat_train = cats[:batch_size-test_size]
cat_test = cats[batch_size-test_size:batch_size]
con_train = conts[:batch_size-test_size]
```





In [111…
```
y_train = y_train.type(torch.LongTensor)
y_test = y_test.type(torch.LongTensor)
```

In [112…
```
import time
start_time = time.time()

epochs = 30
losses = []

for i in range(epochs):
    i+=1
    y_pred = model(cat_train, con_train)
    loss = criterion(y_pred, y_train)
    losses.append(loss)

    if i%25 == 1:
        print(f'epoch: {i:3}  loss: {loss.item():10.8f}')

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

```
epoch:   1  loss: 1.76527536
epoch:  26  loss: 1.47275150
```

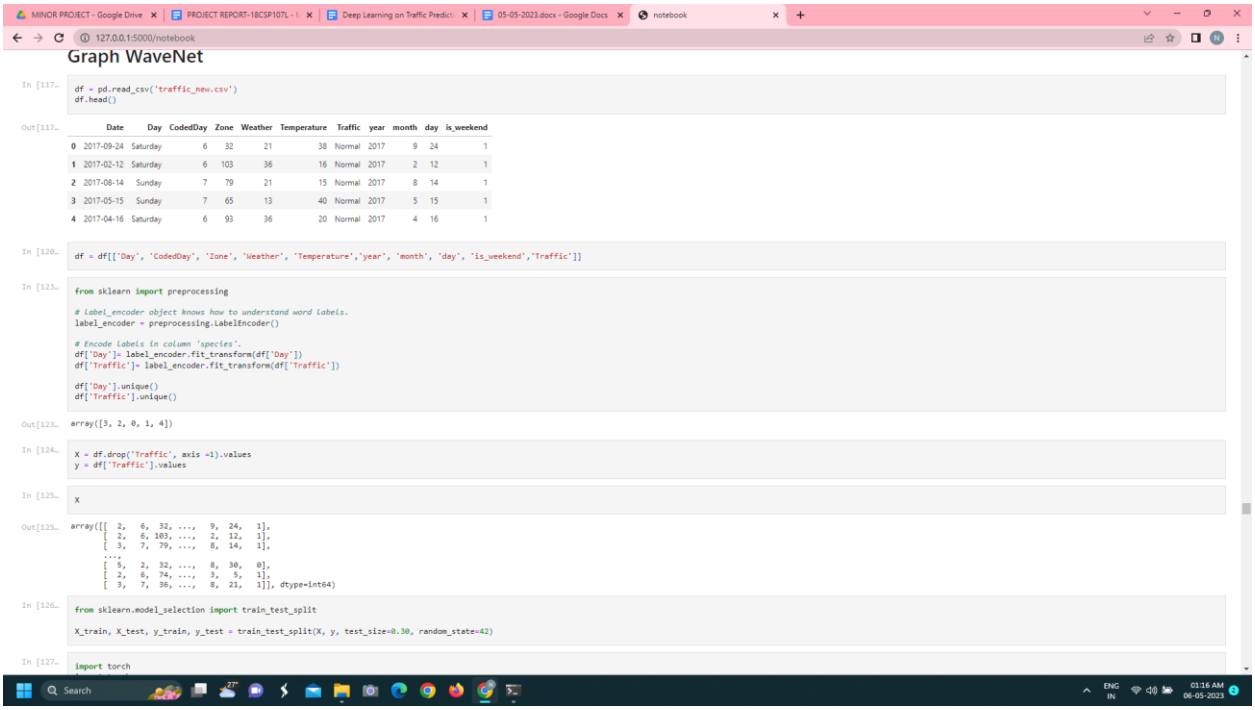
In [113…
```
plt.plot(range(epochs), losses)
plt.ylabel('Cross Entropy Loss')
plt.xlabel('epoch');
```


Graph WaveNet

In [117…
```
df = pd.read_csv('traffic_new.csv')
df.head()
```

Out[117…

	Date	Day	CodedDay	Zone	Weather	Temperature	Traffic	year	month	day	is_weekend
0	2017-09-24	Saturday	6	32	21	38	Normal	2017	9	24	1



Graph WaveNet

```
In [117]:
df = pd.read_csv('traffic_new.csv')
df.head()
```

Out[117]:

	Date	Day	CodedDay	Zone	Weather	Temperature	Traffic	year	month	day	is_weekend
0	2017-09-24	Saturday	6	32	21	38	Normal	2017	9	24	1
1	2017-02-12	Saturday	6	103	36	16	Normal	2017	2	12	1
2	2017-08-14	Sunday	7	79	21	15	Normal	2017	8	14	1
3	2017-05-15	Sunday	7	65	13	40	Normal	2017	5	15	1
4	2017-04-16	Saturday	6	93	36	20	Normal	2017	4	16	1

```
In [120]:
df = df[['Day', 'CodedDay', 'Zone', 'Weather', 'Temperature','year', 'month', 'day', 'is_weekend','Traffic']]
```

```
In [123]:
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
df['Day']= label_encoder.fit_transform(df['Day'])
df['Traffic']= label_encoder.fit_transform(df['Traffic'])

df['Day'].unique()
df['Traffic'].unique()
```

```
Out[123]:
array([3, 2, 0, 1, 4])
```

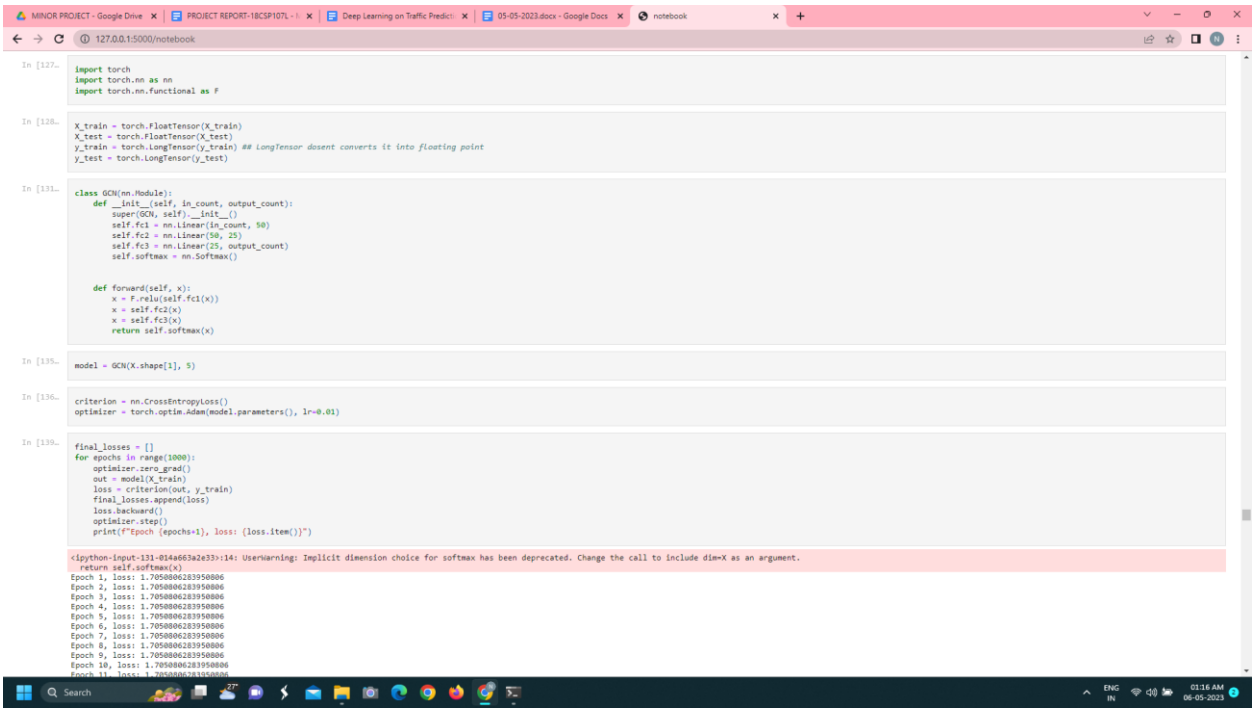
```
In [124]:
X = df.drop('Traffic', axis =1).values
y = df['Traffic'].values
```

```
In [125]:
X
```

```
Out[125]:
array([[ 2,  6, 32, ...,  9, 24,  1],
       [ 2,  6, 103, ...,  2, 12,  1],
       [ 3,  7, 79, ...,  8, 14,  1],
       ...,
       [ 5,  2, 32, ...,  8, 30,  0],
       [ 2,  6, 74, ...,  3,  5,  1],
       [ 3,  7, 36, ...,  8, 21,  1]], dtype=int64)
```

```
In [126]:
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)
```

```
In [127]:
import torch
```



```
In [127]:
import torch
import torch.nn as nn
import torch.nn.functional as F
```

```
In [128]:
X_train = torch.FloatTensor(X_train)
X_test = torch.FloatTensor(X_test)
y_train = torch.LongTensor(y_train) ## LongTensor dosent converts it into floating point
y_test = torch.LongTensor(y_test)
```

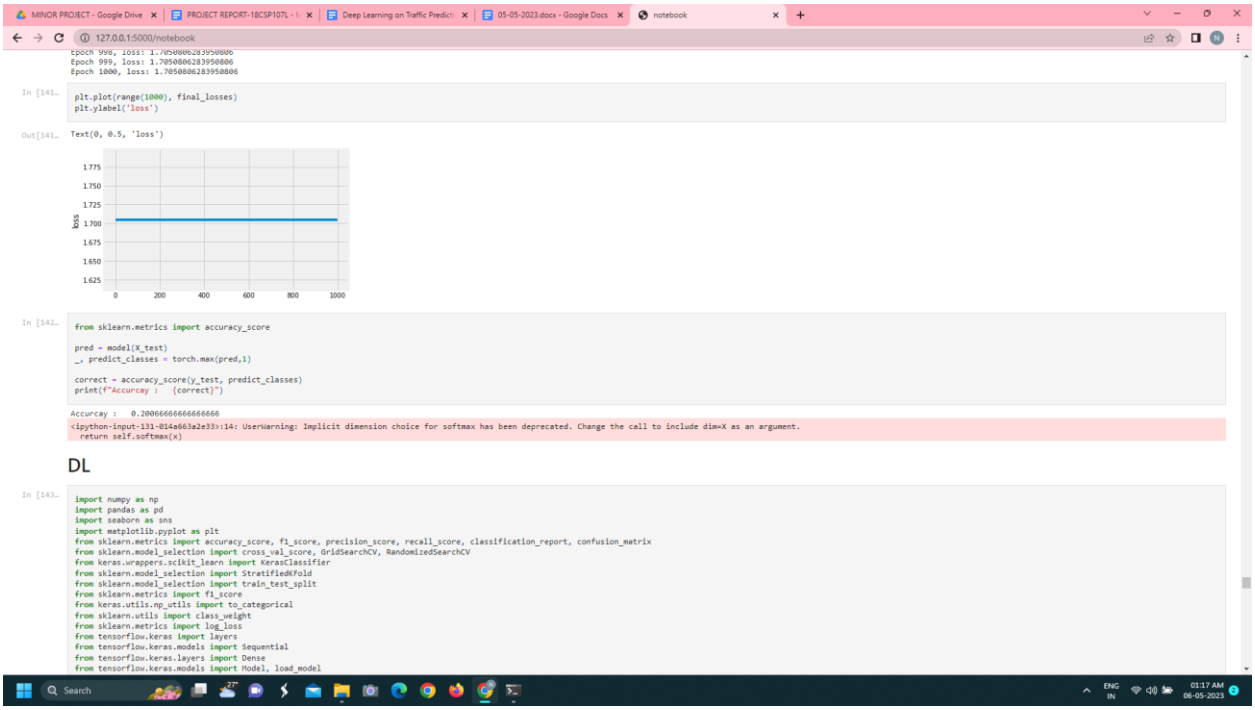
```
In [131]:
class GCN(nn.Module):
    def __init__(self, in_count, output_count):
        super(GCN, self).__init__()
        self.fc1 = nn.Linear(in_count, 50)
        self.fc2 = nn.Linear(50, 25)
        self.fc3 = nn.Linear(25, output_count)
        self.softmax = nn.Softmax()


    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        x = self.fc3(x)
        return self.softmax(x)
```

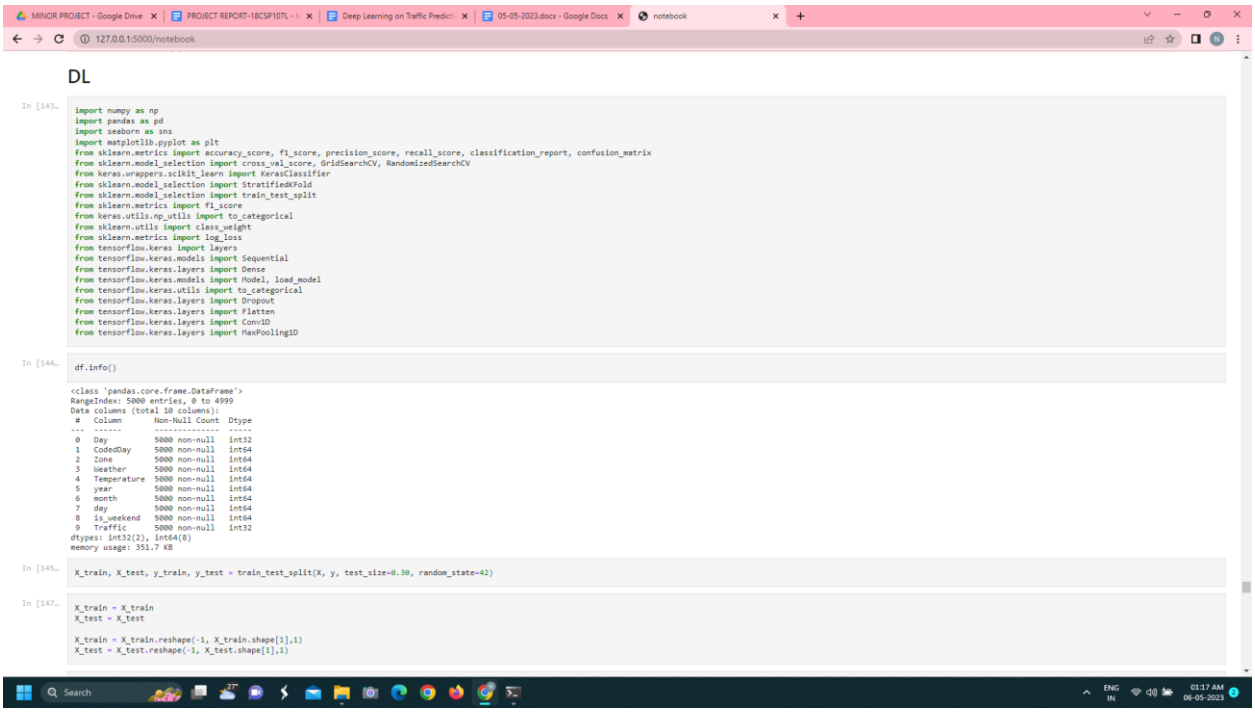
```
In [135]:
model = GCN(X.shape[1], 5)
```

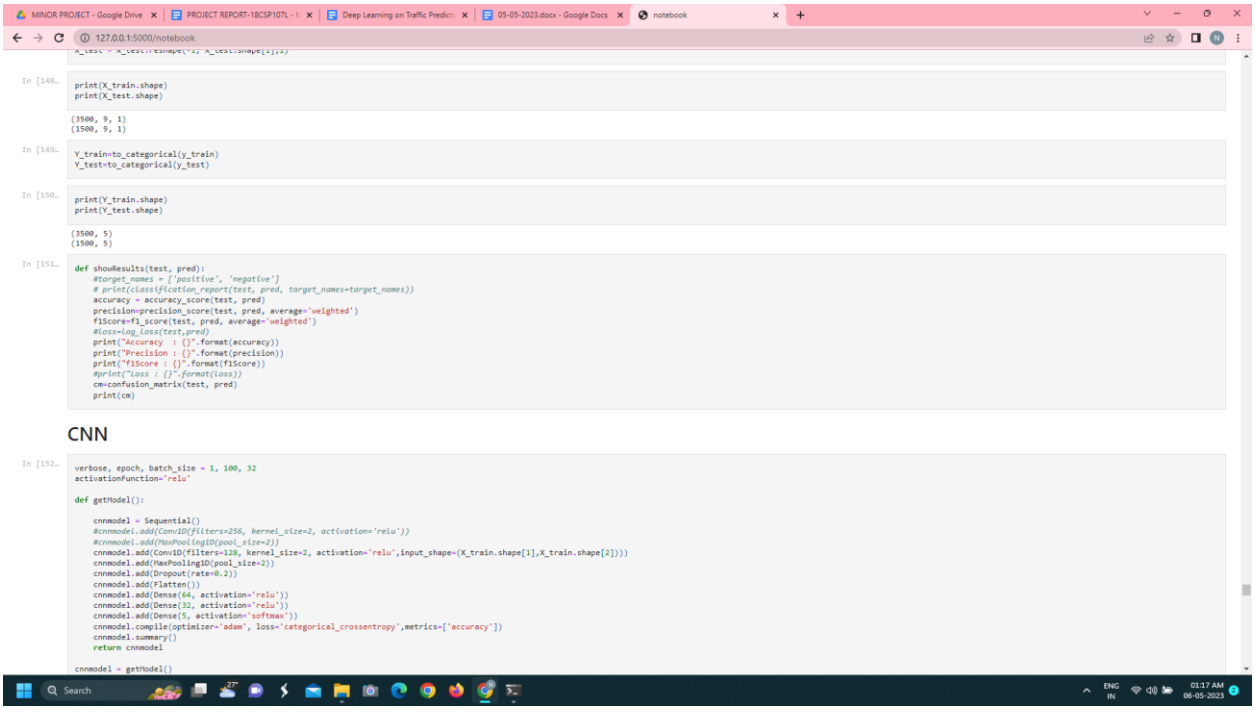
```
In [136]:
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
```

```
In [139]:
final_losses = []
for epochs in range(1000):
    optimizer.zero_grad()
    out = model(X_train)
    loss = criterion(out, y_train)
    final_losses.append(loss)
    loss.backward()
    optimizer.step()
    print(f"Epoch {epochs+1}, loss: {loss.item()}")
```

```
<ipython-input-131-014a663a2e33>:14: UserWarning: Implicit dimension choice for softmax has been deprecated. Change the call to include dim=X as an argument.
  return self.softmax(x)
Epoch 1, loss: 1.7050806283950806
Epoch 2, loss: 1.7050806283950806
Epoch 3, loss: 1.7050806283950806
Epoch 4, loss: 1.7050806283950806
Epoch 5, loss: 1.7050806283950806
Epoch 6, loss: 1.7050806283950806
Epoch 7, loss: 1.7050806283950806
Epoch 8, loss: 1.7050806283950806
Epoch 9, loss: 1.7050806283950806
Epoch 10, loss: 1.7050806283950806
Epoch 11, loss: 1.7050806283950806
```


MINOR PROJECT - Google Drive
PROJECT REPORT-18CSP107L - I
Deep Learning on Traffic Predict
05-05-2023.docx - Google Docs
notebook
127.0.0.1:5000/notebook
Epoch 999, loss: 1.7050806283950806
Epoch 1000, loss: 1.7050806283950806
In [141...
plt.plot(range(1000), final_losses)
plt.ylabel('loss')
Out[141...
Text(0, 0.5, 'loss')
loss
1.775
1.750
1.725
1.700
1.675
1.650
1.625
0
200
400
600
800
1000
In [142...
from sklearn.metrics import accuracy_score
pred = model(X_test)
_, predict_classes = torch.max(pred,1)
correct = accuracy_score(y_test, predict_classes)
print(f"Accuracy : {correct}")
Accuracy : 0.20066666666666666
<ipython-input-131-014a663a2e33>:14: UserWarning: Implicit dimension choice for softmax has been deprecated. Change the call to include dim=X as an argument.
return self.softmax(x)
DL
In [143...
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_report, confusion_matrix
from sklearn.model_selection import cross_val_score, GridSearchCV, RandomizedSearchCV
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
from keras.utils.np_utils import to_categorical
from sklearn.utils import class_weight
from sklearn.metrics import log_loss
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Model, load_model
Search
ENG
IN
01:17 AM
06-05-2023

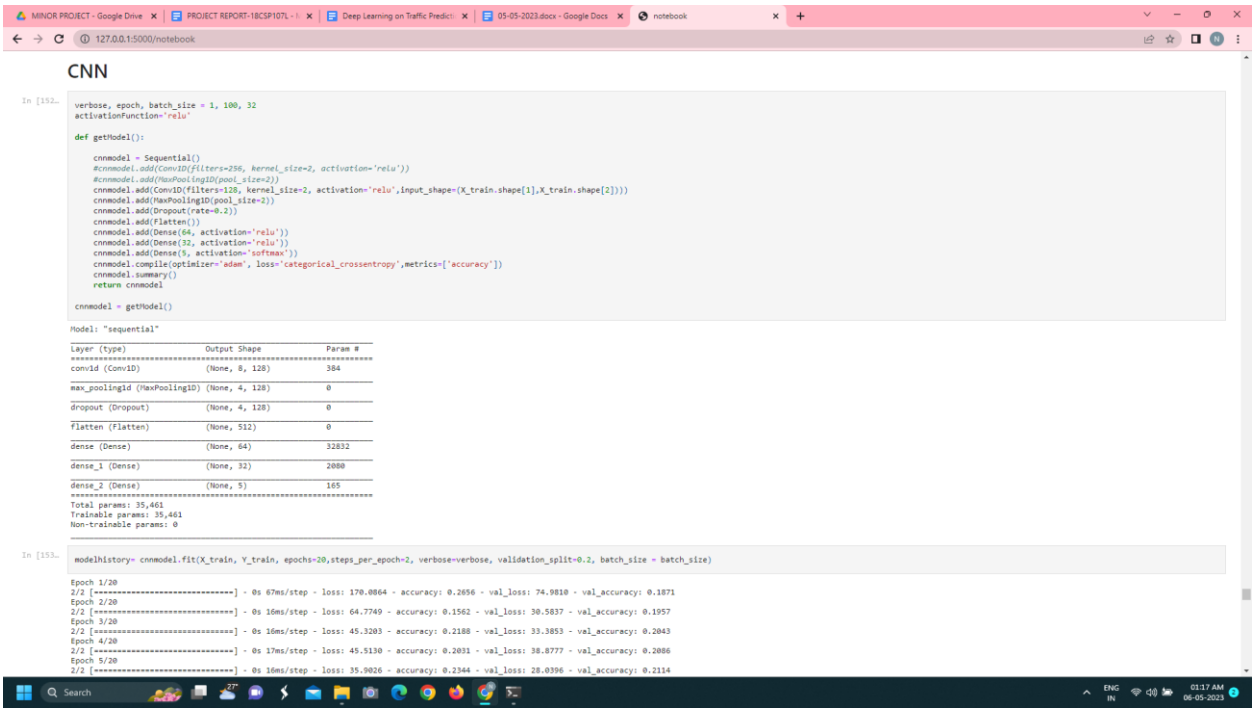

MINOR PROJECT - Google Drive
PROJECT REPORT-18CSP107L - I
Deep Learning on Traffic Predict
05-05-2023.docx - Google Docs
notebook
127.0.0.1:5000/notebook
DL
In [143...
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_report, confusion_matrix
from sklearn.model_selection import cross_val_score, GridSearchCV, RandomizedSearchCV
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
from keras.utils.np_utils import to_categorical
from sklearn.utils import class_weight
from sklearn.metrics import log_loss
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Conv1D
from tensorflow.keras.layers import MaxPooling1D
In [144...
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 10 columns):
Column Non-Null Count Dtype
--- ------ -------------- -----
0 Day 5000 non-null int32
1 CodedDay 5000 non-null int64
2 Zone 5000 non-null int64
3 Weather 5000 non-null int64
4 Temperature 5000 non-null int64
5 year 5000 non-null int64
6 month 5000 non-null int64
7 day 5000 non-null int64
8 is_weekend 5000 non-null int64
9 Traffic 5000 non-null int32
dtypes: int32(2), int64(8)
memory usage: 351.7 KB
In [145...
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)
In [147...
X_train = X_train
X_test = X_test
X_train = X_train.reshape(-1, X_train.shape[1],1)
X_test = X_test.reshape(-1, X_test.shape[1],1)
Search
ENG
IN
01:17 AM
06-05-2023



```
In [148...
print(X_train.shape)
print(X_test.shape)

(3500, 9, 1)
(1500, 9, 1)

In [149...
Y_train=to_categorical(y_train)
Y_test=to_categorical(y_test)

In [150...
print(Y_train.shape)
print(Y_test.shape)

(3500, 5)
(1500, 5)

In [151...
def showResults(test, pred):
    #target_names = ['positive', 'negative']
    # print(classification_report(test, pred, target_names=target_names))
    accuracy = accuracy_score(test, pred)
    precision=precision_score(test, pred, average='weighted')
    f1Score=f1_score(test, pred, average='weighted')
    #loss=log_loss(test,pred)
    print("Accuracy  : {}".format(accuracy))
    print("Precision : {}".format(precision))
    print("f1Score   : {}".format(f1Score))
    #print("Loss : {}".format(loss))
    cm=confusion_matrix(test, pred)
    print(cm)
```

CNN

```
In [152...
verbose, epoch, batch_size = 1, 100, 32
activationFunction='relu'

def getModel():

    cnnmodel = Sequential()
    #cnnmodel.add(Conv1D(filters=256, kernel_size=2, activation='relu'))
    #cnnmodel.add(MaxPooling1D(pool_size=2))
    cnnmodel.add(Conv1D(filters=128, kernel_size=2, activation='relu',input_shape=(X_train.shape[1],X_train.shape[2])))
    cnnmodel.add(MaxPooling1D(pool_size=2))
    cnnmodel.add(Dropout(rate=0.2))
    cnnmodel.add(Flatten())
    cnnmodel.add(Dense(64, activation='relu'))
    cnnmodel.add(Dense(32, activation='relu'))
    cnnmodel.add(Dense(5, activation='softmax'))
    cnnmodel.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'])
    cnnmodel.summary()
    return cnnmodel

cnnmodel = getModel()
```



CNN

```
In [152...
verbose, epoch, batch_size = 1, 100, 32
activationFunction='relu'

def getModel():

    cnnmodel = Sequential()
    #cnnmodel.add(Conv1D(filters=256, kernel_size=2, activation='relu'))
    #cnnmodel.add(MaxPooling1D(pool_size=2))
    cnnmodel.add(Conv1D(filters=128, kernel_size=2, activation='relu',input_shape=(X_train.shape[1],X_train.shape[2])))
    cnnmodel.add(MaxPooling1D(pool_size=2))
    cnnmodel.add(Dropout(rate=0.2))
    cnnmodel.add(Flatten())
    cnnmodel.add(Dense(64, activation='relu'))
    cnnmodel.add(Dense(32, activation='relu'))
    cnnmodel.add(Dense(5, activation='softmax'))
    cnnmodel.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'])
    cnnmodel.summary()
    return cnnmodel

cnnmodel = getModel()
```

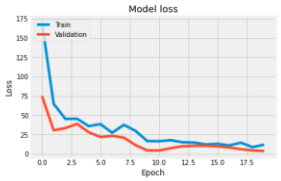
Model: "sequential"

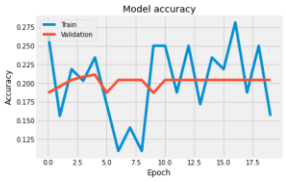
Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 8, 128)	384
max_pooling1d (MaxPooling1D)	(None, 4, 128)	0
dropout (Dropout)	(None, 4, 128)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 64)	32832
dense_1 (Dense)	(None, 32)	2080
dense_2 (Dense)	(None, 5)	165

Total params: 35,461
Trainable params: 35,461
Non-trainable params: 0

```
In [153...
modelhistory= cnnmodel.fit(X_train, Y_train, epochs=20,steps_per_epoch=2, verbose=verbose, validation_split=0.2, batch_size = batch_size)

Epoch 1/20
2/2 [==============================] - 0s 67ms/step - loss: 170.0864 - accuracy: 0.2656 - val_loss: 74.9810 - val_accuracy: 0.1871
Epoch 2/20
2/2 [==============================] - 0s 16ms/step - loss: 64.7749 - accuracy: 0.1562 - val_loss: 30.5837 - val_accuracy: 0.1957
Epoch 3/20
2/2 [==============================] - 0s 16ms/step - loss: 45.3203 - accuracy: 0.2188 - val_loss: 33.3853 - val_accuracy: 0.2043
Epoch 4/20
2/2 [==============================] - 0s 17ms/step - loss: 45.5130 - accuracy: 0.2031 - val_loss: 38.8777 - val_accuracy: 0.2086
Epoch 5/20
2/2 [==============================] - 0s 16ms/step - loss: 35.9026 - accuracy: 0.2344 - val_loss: 28.0396 - val_accuracy: 0.2114
```

```
# Plot training & validation loss values
plt.plot(modelhistory.history['loss'])
plt.plot(modelhistory.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.savefig('loss.png', format='png', dpi=1200)
plt.show()


# Plot training & validation accuracy values
plt.plot(modelhistory.history['accuracy'])
plt.plot(modelhistory.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.savefig('accuracy.png', format='png', dpi=1200)
plt.show()
```






```
cnnpredictions = cnnmodel.predict(X_test, verbose=1)
cnn_predict=np.argmax(cnnpredictions,axis=1)
cnn_actual_value=np.argmax(Y_test,axis=1)
showResults(cnn_actual_value, cnn_predict)
```

```
 1/47 [..............................] - ETA: 0sWARNING:tensorflow:Callbacks method `on_predict_batch_begin` is slow compared to the batch time (batch time: 0.0000s vs `on_predict_batch_begin` time: 0.0010s). Check your callbacks.
47/47 [==============================] - 0s 658us/step
Accuracy  : 0.1933333333333333
Precision : 0.037377777777777774
f1Score : 0.06264432029795158
[[  0   0   0   0 284]
 [  0   0   0   0 301]
 [  0   0   0   0 309]
 [  0   0   0   0 316]
 [  0   0   0   0 290]]
C:\Users\Public\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1221: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

RNN + CNN

```
import tensorflow as tf
tf.keras.backend.clear_session()

model = tf.keras.models.Sequential([tf.keras.layers.Conv1D(filters=64,kernel_size=5,strides=1,padding="causal",activation="relu",input_shape=(X_train.shape[1],X_train.shape[2])),
    tf.keras.layers.MaxPooling1D(pool_size=2, strides=1, padding="valid"),
    tf.keras.layers.Conv1D(filters=32, kernel_size=3, strides=1, padding="causal", activation="relu"),
    tf.keras.layers.MaxPooling1D(pool_size=2, strides=1, padding="valid"),
    tf.keras.layers.SimpleRNN(128, return_sequences=True),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation="relu"),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(32, activation="relu"),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(5)
])

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(5e-4,
                                                             decay_steps=1000000,
                                                             decay_rate=0.98,
                                                             staircase=False)

model.compile(loss=tf.keras.losses.MeanSquaredError(),
              optimizer=tf.keras.optimizers.SGD(learning_rate=lr_schedule, momentum=0.8),
              metrics=['acc'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 9, 64)             384
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 8, 64)             0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 8, 32)             6176
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 7, 32)             0
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 9, 64)             384
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 8, 64)             0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 8, 32)             6176
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 7, 32)             0
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, 7, 128)            20608
_________________________________________________________________
flatten (Flatten)            (None, 896)               0
_________________________________________________________________
dense (Dense)                (None, 128)               114816
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                4128
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 165
=================================================================
Total params: 146,277
Trainable params: 146,277
Non-trainable params: 0
_________________________________________________________________
```

In [157…

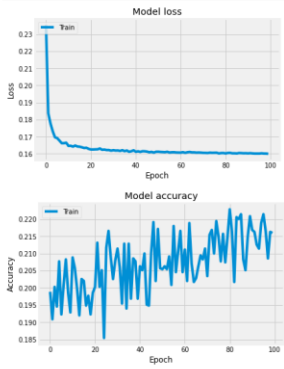
```
history = model.fit(X_train, Y_train,epochs=100)
```

```
Epoch 1/100
110/110 [==============================] - 0s 4ms/step - loss: 0.2353 - acc: 0.1989
Epoch 2/100
110/110 [==============================] - 0s 3ms/step - loss: 0.1837 - acc: 0.1909
Epoch 3/100
110/110 [==============================] - 0s 3ms/step - loss: 0.1777 - acc: 0.2003
Epoch 4/100
110/110 [==============================] - 0s 3ms/step - loss: 0.1732 - acc: 0.1946
Epoch 5/100
110/110 [==============================] - 0s 3ms/step - loss: 0.1696 - acc: 0.2077
Epoch 6/100
110/110 [==============================] - 0s 3ms/step - loss: 0.1692 - acc: 0.1923
Epoch 7/100
110/110 [==============================] - 0s 3ms/step - loss: 0.1677 - acc: 0.2009
Epoch 8/100
110/110 [==============================] - 0s 3ms/step - loss: 0.1662 - acc: 0.2083
Epoch 9/100
110/110 [==============================] - 0s 3ms/step - loss: 0.1662 - acc: 0.1986
Epoch 10/100
110/110 [==============================] - 0s 4ms/step - loss: 0.1665 - acc: 0.1929
Epoch 11/100
110/110 [==============================] - 0s 3ms/step - loss: 0.1646 - acc: 0.2089
Epoch 12/100
110/110 [==============================] - 0s 3ms/step - loss: 0.1647 - acc: 0.2057
Epoch 13/100
110/110 [==============================] - 0s 3ms/step - loss: 0.1642 - acc: 0.1997
Epoch 14/100
110/110 [==============================] - 0s 3ms/step - loss: 0.1648 - acc: 0.1920
```

```
110/110 [==============================] - 0s 3ms/step - loss: 0.1601 - acc: 0.2163
Epoch 100/100
110/110 [==============================] - 0s 4ms/step - loss: 0.1600 - acc: 0.2160
```

In [158…

```
# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.savefig('loss.png', format='png', dpi=1200)
plt.show()


# Plot training & validation accuracy values
plt.plot(history.history['acc'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.savefig('accuracy.png', format='png', dpi=1200)
plt.show()
```

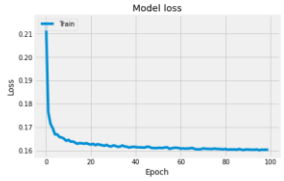


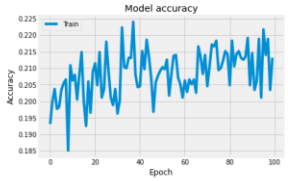
DCRNN - Disfusion CNN with RNN and GRU

```
import tensorflow as tf
tf.keras.backend.clear_session()

model = tf.keras.models.Sequential([tf.keras.layers.Conv1D(filters=64,kernel_size=5,strides=1,padding="causal",activation="relu",input_shape=[X_train.shape[1],X_train.shape[2]]),
    tf.keras.layers.MaxPooling1D(pool_size=2, strides=1, padding="valid"),
    tf.keras.layers.Conv1D(filters=32, kernel_size=3, strides=1, padding="causal", activation="relu"),
    tf.keras.layers.MaxPooling1D(pool_size=2, strides=1, padding="valid"),
    tf.keras.layers.GRU(128, return_sequences=True),
    tf.keras.layers.SimpleRNN(128, return_sequences=True),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation="relu"),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(32, activation="relu"),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(5)
])

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(5e-4,
                                                             decay_steps=1000000,
                                                             decay_rate=0.98,
                                                             staircase=False)

model.compile(loss=tf.keras.losses.MeanSquaredError(),
              optimizer=tf.keras.optimizers.SGD(learning_rate=lr_schedule, momentum=0.8),
              metrics=['acc'])
model.summary()
```

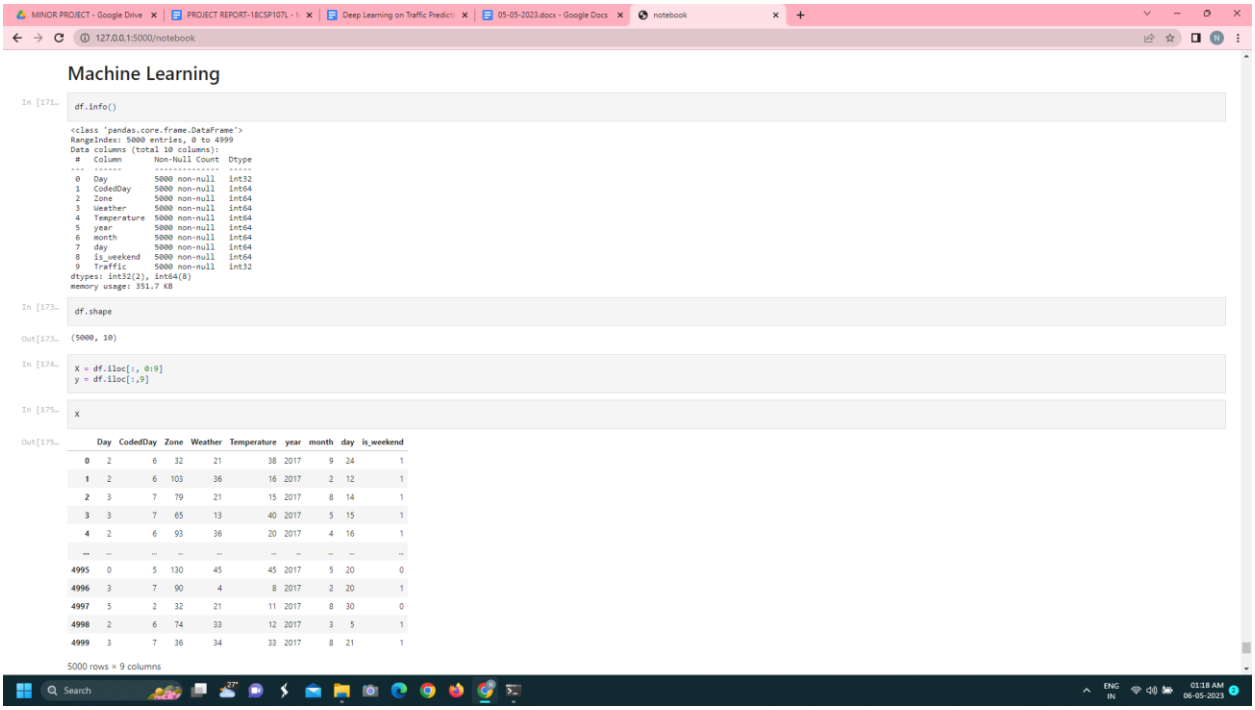
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 9, 64)             384
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 8, 64)             0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 8, 32)             6176
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 7, 32)             0
_________________________________________________________________
gru (GRU)                    (None, 7, 128)            62208
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, 7, 128)            32896
_________________________________________________________________
flatten (Flatten)            (None, 896)               0
_________________________________________________________________
dense (Dense)                (None, 128)               114816
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                4128
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 165
=================================================================
Total params: 220,773
Trainable params: 220,773
```

```
# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.savefig('loss.png', format='png', dpi=1200)
plt.show()


# Plot training & validation accuracy values
plt.plot(history.history['acc'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.savefig('accuracy.png', format='png', dpi=1200)
plt.show()
```





Machine Learning



Machine Learning

In [171…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Day          5000 non-null   int32
 1   CodedDay     5000 non-null   int64
 2   Zone         5000 non-null   int64
 3   Weather      5000 non-null   int64
 4   Temperature  5000 non-null   int64
 5   year         5000 non-null   int64
 6   month        5000 non-null   int64
 7   day          5000 non-null   int64
 8   is_weekend   5000 non-null   int64
 9   Traffic      5000 non-null   int32
dtypes: int32(2), int64(8)
memory usage: 351.7 KB
```

In [173…
```
df.shape
```

Out[173… `(5000, 10)`

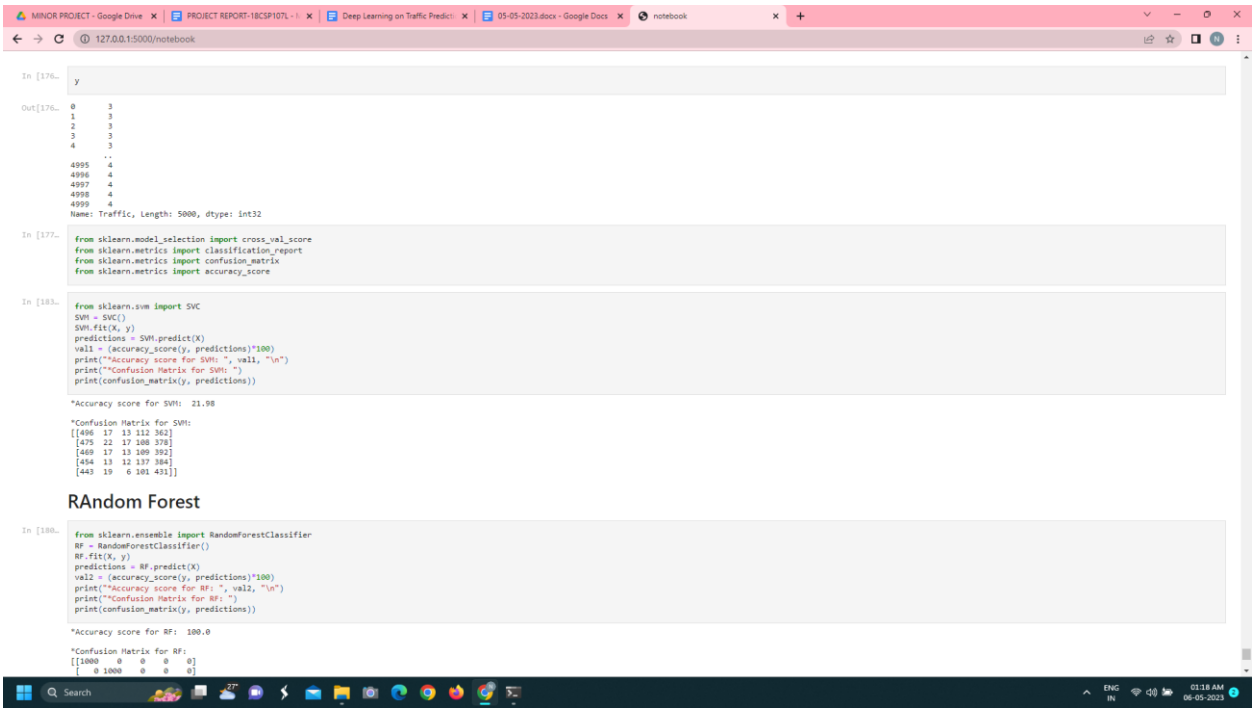
In [174…
```
X = df.iloc[:, 0:9]
y = df.iloc[:,9]
```

In [175…
```
X
```

Out[175…

	Day	CodedDay	Zone	Weather	Temperature	year	month	day	is_weekend
0	2	6	32	21	38	2017	9	24	1
1	2	6	103	36	16	2017	2	12	1
2	3	7	79	21	15	2017	8	14	1
3	3	7	65	13	40	2017	5	15	1
4	2	6	93	36	20	2017	4	16	1
…	…	…	…	…	…	…	…	…	…
4995	0	5	130	45	45	2017	5	20	0
4996	3	7	90	4	8	2017	2	20	1
4997	5	2	32	21	11	2017	8	30	0
4998	2	6	74	33	12	2017	3	5	1
4999	3	7	36	34	33	2017	8	21	1

5000 rows × 9 columns



In [176…
```
y
```

Out[176…
```
0       3
1       3
2       3
3       3
4       3
       ..
4995    4
4996    4
4997    4
4998    4
4999    4
Name: Traffic, Length: 5000, dtype: int32
```

In [177…
```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
```

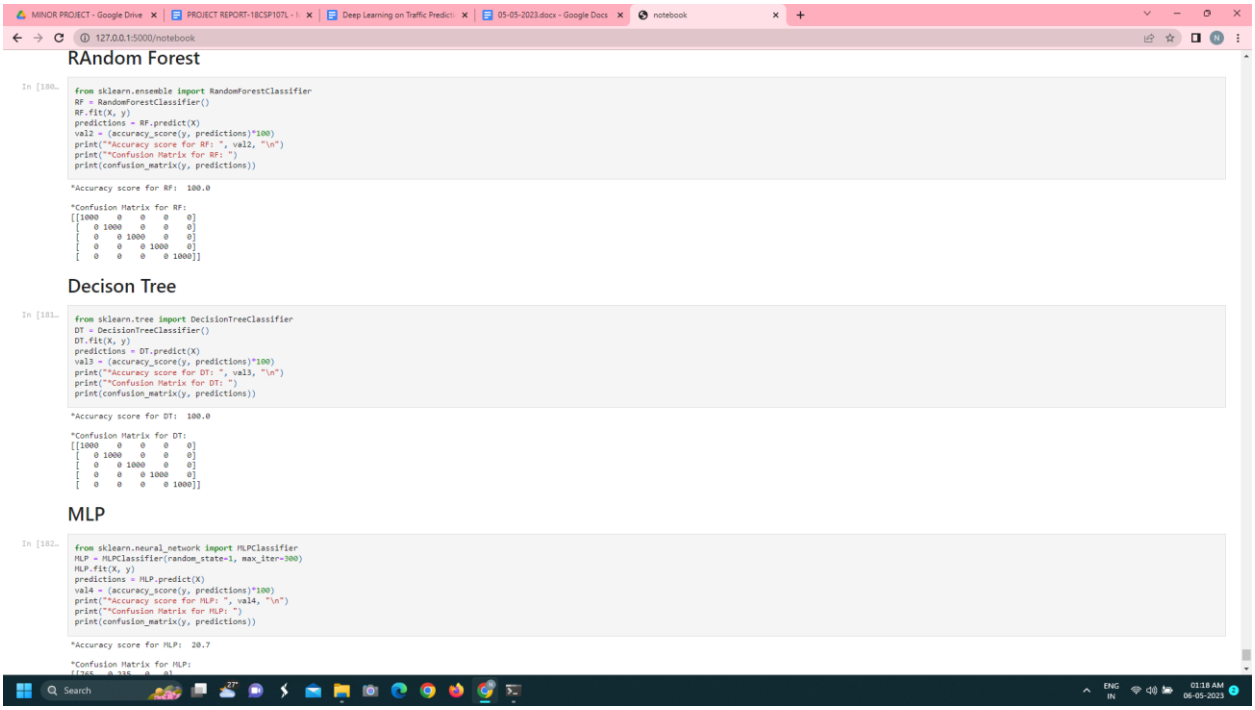
In [183…
```
from sklearn.svm import SVC
SVM = SVC()
SVM.fit(X, y)
predictions = SVM.predict(X)
val1 = (accuracy_score(y, predictions)*100)
print("Accuracy score for SVM: ", val1, "\n")
print("Confusion Matrix for SVM: ")
print(confusion_matrix(y, predictions))
```

```
Accuracy score for SVM:  21.98

Confusion Matrix for SVM:
[[496  17  13 112 362]
 [475  22  17 108 378]
 [469  17  13 109 392]
 [454  13  12 137 384]
 [443  19   6 101 431]]
```

RAndom Forest

In [180…
```
from sklearn.ensemble import RandomForestClassifier
RF = RandomForestClassifier()
RF.fit(X, y)
predictions = RF.predict(X)
val2 = (accuracy_score(y, predictions)*100)
print("Accuracy score for RF: ", val2, "\n")
print("Confusion Matrix for RF: ")
print(confusion_matrix(y, predictions))
```

```
Accuracy score for RF:  100.0

Confusion Matrix for RF:
[[1000    0    0    0    0]
 [   0 1000    0    0    0]
```



RAndom Forest

```
from sklearn.ensemble import RandomForestClassifier
RF = RandomForestClassifier()
RF.fit(X, y)
predictions = RF.predict(X)
val2 = (accuracy_score(y, predictions)*100)
print("Accuracy score for RF: ", val2, "\n")
print("Confusion Matrix for RF: ")
print(confusion_matrix(y, predictions))
```

```
Accuracy score for RF:  100.0

Confusion Matrix for RF:
[[1000    0    0    0    0]
 [   0 1000    0    0    0]
 [   0    0 1000    0    0]
 [   0    0    0 1000    0]
 [   0    0    0    0 1000]]
```

Decison Tree

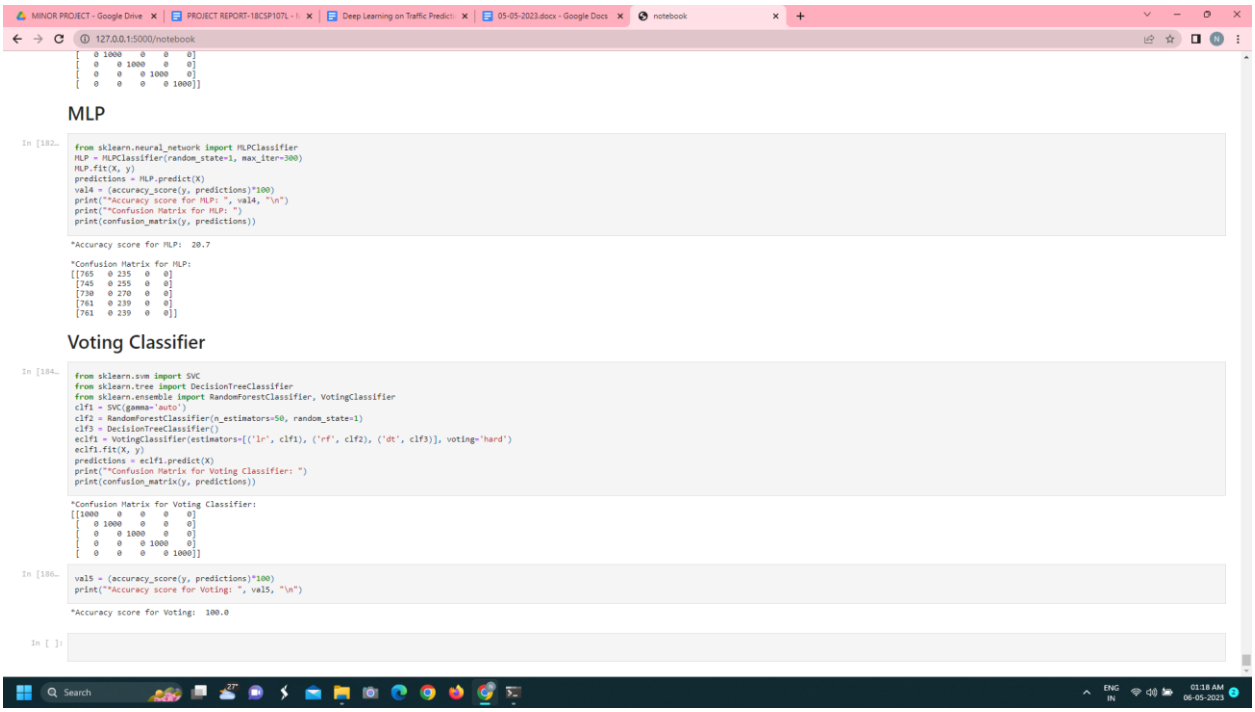
```
from sklearn.tree import DecisionTreeClassifier
DT = DecisionTreeClassifier()
DT.fit(X, y)
predictions = DT.predict(X)
val3 = (accuracy_score(y, predictions)*100)
print("Accuracy score for DT: ", val3, "\n")
print("Confusion Matrix for DT: ")
print(confusion_matrix(y, predictions))
```

```
Accuracy score for DT:  100.0

Confusion Matrix for DT:
[[1000    0    0    0    0]
 [   0 1000    0    0    0]
 [   0    0 1000    0    0]
 [   0    0    0 1000    0]
 [   0    0    0    0 1000]]
```

MLP

```
from sklearn.neural_network import MLPClassifier
MLP = MLPClassifier(random_state=1, max_iter=300)
MLP.fit(X, y)
predictions = MLP.predict(X)
val4 = (accuracy_score(y, predictions)*100)
print("Accuracy score for MLP: ", val4, "\n")
print("Confusion Matrix for MLP: ")
print(confusion_matrix(y, predictions))
```

```
Accuracy score for MLP:  20.7

Confusion Matrix for MLP:
```




```
[   0 1000    0    0    0]
 [   0    0 1000    0    0]
 [   0    0    0 1000    0]
 [   0    0    0    0 1000]]
```

MLP

```
from sklearn.neural_network import MLPClassifier
MLP = MLPClassifier(random_state=1, max_iter=300)
MLP.fit(X, y)
predictions = MLP.predict(X)
val4 = (accuracy_score(y, predictions)*100)
print("Accuracy score for MLP: ", val4, "\n")
print("Confusion Matrix for MLP: ")
print(confusion_matrix(y, predictions))
```

```
Accuracy score for MLP:  20.7

Confusion Matrix for MLP:
[[765   0 235   0   0]
 [745   0 255   0   0]
 [730   0 270   0   0]
 [761   0 239   0   0]
 [761   0 239   0   0]]
```


Voting Classifier

```
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
clf1 = SVC(gamma='auto')
clf2 = RandomForestClassifier(n_estimators=50, random_state=1)
clf3 = DecisionTreeClassifier()
eclf1 = VotingClassifier(estimators=[('lr', clf1), ('rf', clf2), ('dt', clf3)], voting='hard')
eclf1.fit(X, y)
predictions = eclf1.predict(X)
print("Confusion Matrix for Voting Classifier: ")
print(confusion_matrix(y, predictions))
```

```
Confusion Matrix for Voting Classifier:
[[1000    0    0    0    0]
 [   0 1000    0    0    0]
 [   0    0 1000    0    0]
 [   0    0    0 1000    0]
 [   0    0    0    0 1000]]
```

```
val5 = (accuracy_score(y, predictions)*100)
print("Accuracy score for Voting: ", val5, "\n")
```

```
Accuracy score for Voting:  100.0
```

PAPER PUBLICATIONS

Paper not submitted

Deep Learning on Traffic Prediction Methods Analysis and Future Directions

Nurukurthi Veera Venkata Ganesh
Department of Data Science and Business Systems,
School of Computing,
SRM Institute of Science and Technology,
Kattankulathur, Tamil Nadu, India

Sri Surya Hemanth
Department of Data Science and Business Systems,
School of Computing,
SRM Institute of Science and Technology,
Kattankulathur, Tamil Nadu, India

Dr.T.Veeramakali
Department of Data Science and Business Systems,
School of Computing,
SRM Institute of Science and Technology,
Kattankulathur, Tamil Nadu, India
veeramat@srmist.edu.in

ABSTRACT: Accurate traffic forecasting is essential for intelligent transportation frameworks. Course arranging, vehicle dispatch, and facilitating gridlock may all profit from precise traffic anticipating. This challenge is difficult due to the complex and dynamic spatiotemporal relationships that exist between various segments of the road network. This topic has recently received a lot of research attention, particularly the deep learning method, which has significantly improved traffic forecasting abilities. This work expects to introduce a far reaching assessment of traffic prediction strategies in view of deep gaining according to different points of view. To get things started, we'll summarize and classify the current methods for traffic forecasting. Second, we discuss the most recent approaches utilized in various applications for traffic prediction. Thirdly, to help different specialists, we gather and organize generally utilized public datasets from the ongoing writing. In addition, we conduct extensive investigations to evaluate the display of various methodologies on a genuine public dataset in order to provide an evaluation and examination. In conclusion, we examine irritating issues in this area.

Keywords *– Traffic Prediction, Deep Learning, Spatial Temporal Dependency Modeling.*

1. INTRODUCTION

A brilliant city is gradually taking the place of the cutting-edge city. The rapid rate of population growth and urbanization puts a serious strain on traffic across the board in metropolitan areas. The Intelligent Transportation System (ITS), which is an essential component of successful urban communities, includes traffic anticipation as one of its fundamental components. Precise traffic guaging is essential for some genuine applications. Traffic stream expectation, for example, may assist urban communities with eliminating clog; Estimates of vehicle hailing request can inspire vehicle sharing organizations to pre-assign vehicles to regions with popularity. We can look at this topic from different perspectives now that more information about traffic is available.

PLAGIARISM REPORT

Traffic detection using dep learning

ORIGINALITY REPORT

10%	2%	1%	10%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Clarkston Community Schools Student Paper	9%
2	www.ukessays.com Internet Source	<1%
3	Submitted to University of West London Student Paper	<1%
4	Submitted to VNR Vignana Jyothi Institute of Engineering and Technology Student Paper	<1%
5	Submitted to City University Student Paper	<1%
6	ijisae.org Internet Source	<1%
7	Ortiz Soto Brenda. "Ejercicios introductorios a la programación en Python para el análisis de datos geológicos georreferenciados", TESIUNAM, 2022 Publication	<1%
8	basicedu.uodiyala.edu.iq Internet Source	<1%

9	dokumen.pub Internet Source	<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		