MOTION PREDICTION CHALLENGE ON WAYMO OPEN MOTION DATASET

Thesis

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE AND INFORMATION SECURITY

by

Gudiwada Raghava

(Reg. No.: 2120280)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA

SURATHKAL, MANGALORE - 575025

MAY, 2023

DECLARATION

I hereby declare that the P.G **Project Work Thesis** entitled **Motion Prediction Challenge on Waymo Open Motion Dataset** which is being submitted to the National Institute of Technology Karnataka Surathkal, in partial fulfilment of the requirements for the award of the Degree of **Master of Technology in Computer Science and Engineering-Information Security** in the department of **Computer Science and Engineering**, is a bonafide report of the work carried out by me. The material contained in this Report has not been submitted to any University or Institution for the award of any degree.

Gudiwada Raghava
212IS009
Department of Computer Science and Engineering
NITK, Surathkal

Place: NITK, Surathkal.
Date: Friday 26th May, 2023

CERTIFICATE

This is to certify that the P.G. **Project Work Thesis** entitled **Motion Prediction Challenge on Waymo Open Motion Dataset** submitted by **Gudiwada Raghava 212IS009** as the record of the work carried out by her, is accepted as the P.G. Project Work Thesis submission in partial fulfilment of the requirements for the award of the degree **Master of Technology in Computer Science and Engineering-Information Security** in the Department of **Computer Science and Engineering**.

Guide
Dr. Basavaraj Talawar
Department of Computer Science and Engineering
NITK, Surathkal

Chairman - DPGC
Dr. Manu Basavaraju
Department of Computer Science and Engineering
NITK, Surathkal

ACKNOWLEDGEMENT

I take this opportunity to express a deep sense of gratitude towards my guide **Dr. Basavaraj Talawar**, for providing excellent guidance, encouragement, and inspiration throughout the project work. Without her invaluable guidance, this work would never have been successful. I would also like to thank all my teammates for their valuable suggestions and helpful discussions. Thank you for providing such an opportunity.

Gudiwada Raghava (212IS009)

NIT Karnataka

Date: Friday 26th May, 2023

Abstract

One of the most difficult and crucial issues with autonomous driving is predicting the future behavior of other drivers. In order to apply deep learning to this issue, rich perceptual signals and map data must be combined, and it is necessary to infer extremely multi-modal distributions over conceivable futures. The field of autonomous driving is evolving quickly, and today the first driver-less rides are being offered in urban settings. The technology must meet the highest requirements for safety and dependability. The motion prediction component of the overall self-driving pipeline is essential in delivering these attributes. Recently, motion prediction is receiving increasing attention as it is crucial for autonomous vehicles to make safe decisions. It is also a highly challenging task due to the inherently multi-modal behaviors of the agent and complex scene environments.

Motor vehicle accidents are a significant cause of death and autonomous vehicles (AVs) have been suggested as a potential solution in reducing accidental deaths. However, a significant challenge to the adoption of autonomous vehicles remains motion prediction. Our work leverages the Waymo Open Dataset is a collection of scenarios containing "critical scenarios" that demand complex interaction modeling with information on object states and roadgraph features. We develop a two-step trajectory prediction model that first predicts the goals of target objects, then uses these goals as anchors to complete their trajectories. We found our results to affirm the potential of modular approaches in motion prediction tasks, achieving decent performance in goal prediction and high performance in trajectory completion module.

Keywords Motion prediction, Autonomous driving vehicles, Trajectory Prediction, Goal Prediction, and Rasterisation.

Contents

List of Figures

Chapter 1

Introduction

The World Health Organization estimates that 1.3 million people die from traffic accidents every year *World Health Organization. Road traffic injuries.* (2021). A majority of these accidents are avoidable, a product of human error. With the recent development of autonomous vehicles (AVs), it seems possible to drastically reduce the number of accidents and save lives. In order to do so, and to be able to improve the safety of AVs, one important task that has attracted a lot of attention is to be able to accurately predict the motion of nearby objects. This project aims to predict the trajectories of vehicles, pedestrians, and cyclists (referred to as 'objects') up to 8 seconds into the future provided with one second of context. The context includes information about the current scene such as the coordinates of streets and lanes, the states and locations of traffic lights and signs, and the positions, velocities, and directions of all objects in the scene. In our project, we build a tractable model that can tackle the problem of motion prediction in critical situations. To do so, we leverage a modular approach with two steps, first predicting the desired endpoints for all objects, then estimating the trajectory conditioned on these end points.

Road accident fatalities are primarily caused by human mistake, which includes distraction, weariness, disobedience to driving laws, and bad judgement (SAE (2018)). An excellent potential to lower these errors and thereby raise road safety, accident costs, productivity, mobility, and convenience is provided by automating the driving task. Globally, Automated Driving Systems (ADS) are being researched intensively in order to realise these enormous potentials. ADS (SAE level 3, 4, or 5) can manage both lateral and longitudinal vehicle motions at once (SAE (2018)).

Well-designed performance measurements must be realistic and, ideally, free of arbitrary terms (such thresholds). Supporting data from simulations and field tests is required as safety is being measured. It is premature to regulate the safety requirement for ADS NHTSA (2020) claims the National Highway Traffic Safety Administration (NHTSA). The researchers and ADS developers cannot agree on any specific performance measurements. Performance measurements that are poorly designed (as opposed to those that are effectively designed) may impede the onset or progression of ADS or, worse still, provide one a false sense of security or performance. Therefore, in order to create the safety criteria for ADS, NHTSA is currently looking for feedback from researchers, authorities, and ADS developers. The World Forum for Harmonisation of Vehicle Regulations, which represents the European Union, is also actively engaged in this issue. This emphasises the necessity of conducting a literature evaluation of the performance indicators available to evaluate the effectiveness of ADS.

The comprehension of driving tasks is necessary for the formulation of performance measurements. Human drivers' ability to execute driving activities safely is dependent on

1. knowing the current state of self (such as acceleration, location, steering angle, and speed).
2. perceiving the states of surrounding obstacles
3. planning the future course of action ensuring safety, and
4. controlling the vehicle using steering wheel, throttle, and brakes. Analogously, ADS can be considered to have four primary modules (Figure 1.1):
    - Localization
    - Vehicle control
    - Motion planning, and
    - Perception.

ADS can also have an additional module dedicated to wireless communications.

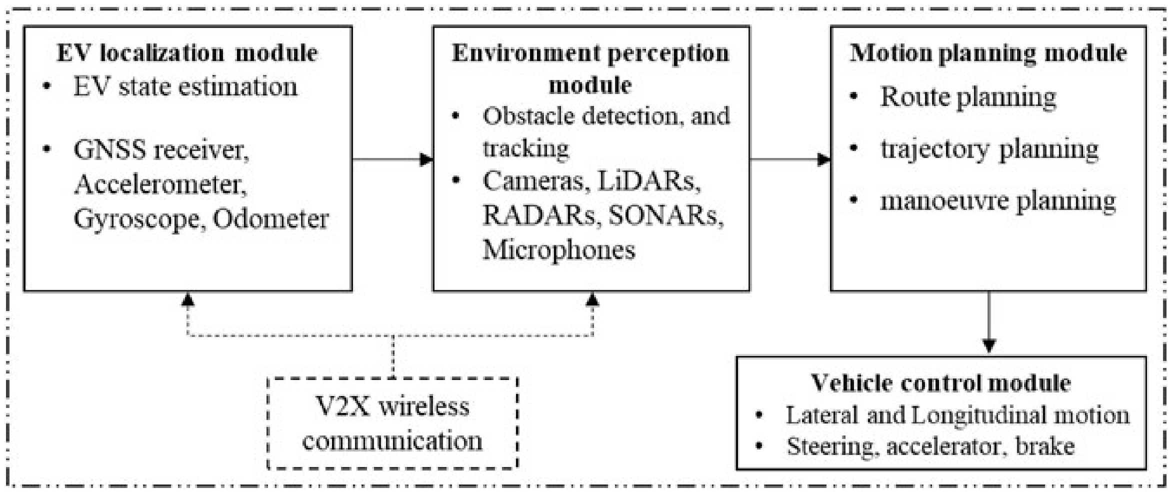

EV localization module
• EV state estimation
• GNSS receiver, Accelerometer, Gyroscope, Odometer
Environment perception module
• Obstacle detection, and tracking
• Cameras, LiDARs, RADARs, SONARs, Microphones
Motion planning module
• Route planning
• trajectory planning
• manoeuvre planning
V2X wireless communication
Vehicle control module
• Lateral and Longitudinal motion
• Steering, accelerator, brake

Figure 1.1: Primary modules in an ADS.

Measurements of an EV's status, like as position, velocity, and acceleration, are required for ego vehicle (EV)localization. For a preliminary estimate of the state, one can utilize global navigation satellite systems (GNSS) like GPS, Galileo, GLONASS, and BeiDou or regional navigation satellite systems (RNSS)like IRNSS or QZSS. Such independent systems' localization accuracy is typically unsuitable for applications that are vital to human safety. The localization accuracy can be improved by combining GNSS/RNSS location data with GIS road maps and other sensors like accelerometers and gyroscopes (Yang & Deng (2018)). However, in tunnels and in areas with dense vegetation, GNSS/RNSS accessibility is reduced. Multi-path errors are brought about by urbanization, which could lower the accuracy of localization. It is also possible to localize lanes using visual signals like lane markings or other traffic indicators (Kim & Yi (2017)). But occlusion can cause errors in such systems.

Given a road map and the historical trajectories of the observed agents, a motion prediction model forecasts the future trajectories of the agents of interest. Based on the encoding format of the map and history trajectories, motion prediction methods may be categorized as raster-based, vector-based and graph-based. While the other two might involve transformers or other vector operations to encode the structured road and agent properties, the raster-based approaches often use CNN to encrypt the spatial information. The raster-based models may have more limitations on the range scope (due to memory constraints) and implicit encoding of agent attributes (e.g., velocity), but the CNN structure has been well established and may be preferred by some deep learning accelerators. The vector/graph-based models, on the other hand, are usually featured with a lighter memory footprint, larger receptive fields, and explicit encoding of agent's attributes, forming a more promising methodology.

The most common methodology for developing autonomous driving technology involves several steps: collecting data from sensors, resolving perception issues to identify nearby objects, localization, motion prediction, and lastly motion planning. We only address the motion prediction problem in this study, using the relative positions of other agents and road lines as our only source of ground truth. The motion prediction problem has been addressed in a number of publications, but it has remained a difficult and complex topic to date. The conduct of other agents is naturally unpredictable, which is one of the major challenges. In this research, we propose a multi-path-based methodBalakrishnan Varadarajan & Sapp. (2021).

From an engineering point of view, if the range scope is small, e.g., for pedestrians and cyclists whose velocity is much smaller than vehicles, the raster-based methods maybe much more affordable and can benefit from well established CNN networks and hardware. However, for vehicles, it makes much more sense to utilize vector or graph-based methods. Considering both, we design a general motion prediction framework that may be an ensemble of one or both of these models, aiming to take advantage of both to achieve better prediction accuracy.

One of the key components of a self-driving system is motion prediction Moritz Werling & Thrun. (2010)-Sergio Casas & Urtasun (2017). An autonomous vehicle (AV) must be able to forecast other traffic agents' future trajectories with accuracy, including those of other vehicles, cyclists, and pedestrians. However, it is still quite difficult to find solutions for arbitrary environment scenarios for future motion prediction and the AV's route planning. In this study, we tackle the motion prediction task. The most prominent approaches include image-based models which leverage birds-eye-view rasterised scene representations Henggang Cui & Djuric. (2019)-Namhoon Lee

& Chandraker. (2017) and methods incarnated using graph neural networks.

We establish a simple yet efficient motion prediction baseline based purely on Convolutional Neural Networks (CNN). Our model takes a raster image centered around a target agent as input and directly predicts a set of possible trajectories along with their confidence. The raster image is obtained by the rasterisation of a scene and history of the all the agents. We evaluate our model for the 2022 Waymo Open Dataset Motion Prediction Challenge Scott Ettinger (2021).

To gather a massive dataset, Waymo deployed a fleet of vehicles fitted with LIDARs and cameras. They were cruising Palo Alto's streets while they observed their surroundings.

Using a fleet of vehicles outfitted with LIDARs and cameras, Autonomous driving vehicles gathered a sizable amount of data as seen in figures 1.2 & 1.4. They were cruising Palo Alto's streets while they observed their surroundings.

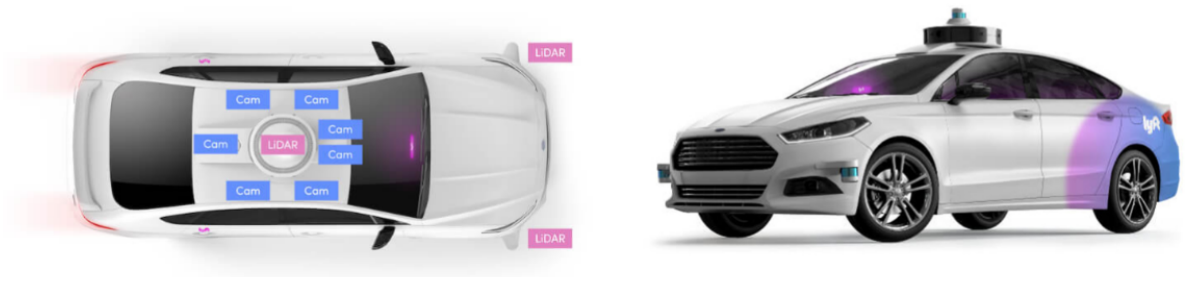


Figure 1.2: A self-driving vehicle that was utilised to gather the data

LIDARs & cameras were used to create 3D point clouds, which were then processed to find every object in the area (other automobiles, road signs, bicycles, pedestrians, and so on). Then, Lyft developers used GPS and the accompanying 3D coordinates to record things on the map as shown in figure 1.3.

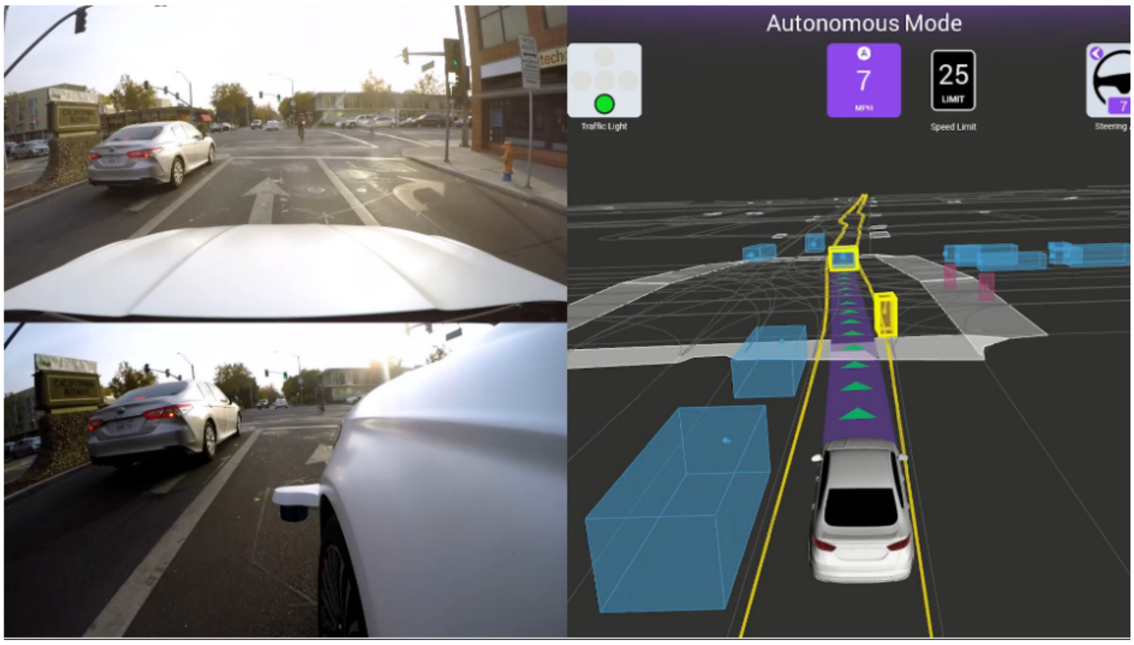


Figure 1.3: (Right)Visualization of perception system, (Left)Images Captured by camera.

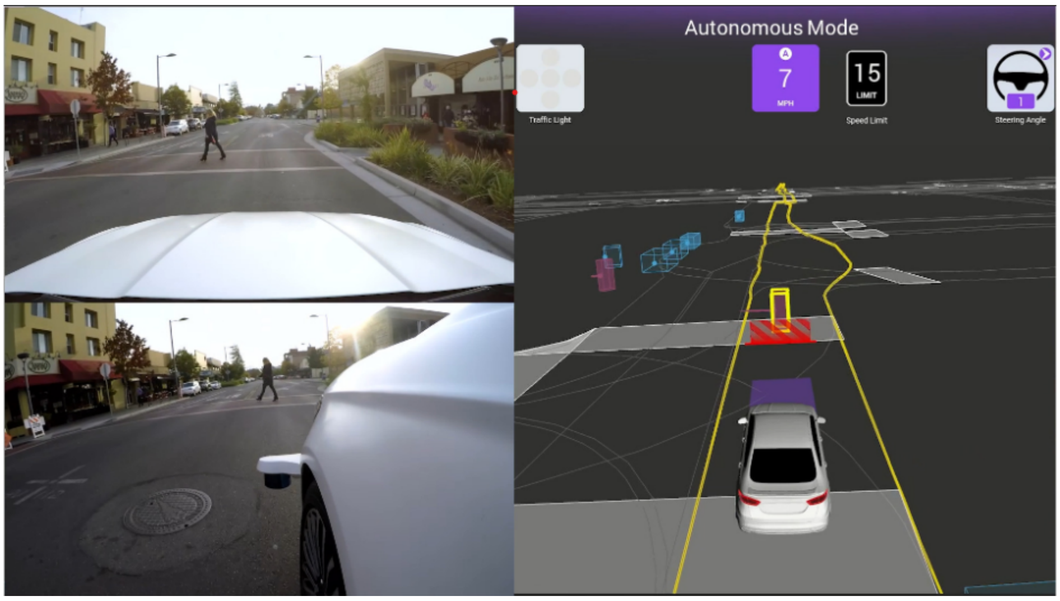


Figure 1.4: Collection of Dataset.

To cover a wide range of road conditions, they record a lot of journeys. There are over 1000 hours of driving altogether in the gathered dataset H. Cui & Djuric (2019).

The perception task is essentially solved, as seen in the figure 1.5, and the results produced by the existing Lyft pipeline are comparatively solid. The correct planning of an autonomous vehicle's (AV) future operations, however, it must be aware of adjacent agents (e.g., other pedestrians, cyclists, and cars,).

“Future motion prediction and the AV’s route planning are extremely difficult challenges that have not yet been resolved for the general situation.”

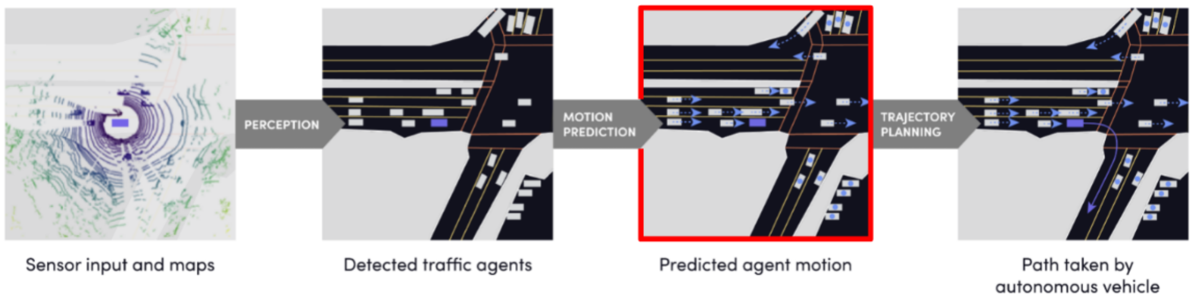


Figure 1.5: From sensory input to path planning

The Waymo Motion Prediction Challenge represents a significant milestone in the advancement of self-driving technology. As the field of autonomous vehicles continues to evolve, accurate prediction of the behavior of other road users becomes crucial for ensuring safe and efficient navigation. Waymo, a leading company in the self-driving industry, organized this challenge to foster innovation and push the boundaries of motion prediction algorithms.

The primary objective of the Waymo Motion Prediction Challenge is to tackle the complex task of predicting the future trajectories of various objects on the road, including vehicles, cyclists, pedestrians, and other dynamic entities. By accurately forecasting the intentions and movements of these entities, autonomous vehicles can make informed decisions and proactively adapt to changing traffic scenarios.

The challenge brings together researchers, engineers, and data scientists from diverse backgrounds to develop novel approaches and algorithms capable of robustly predicting motion in complex driving scenarios. Participants are presented with a large-scale dataset collected by Waymo’s fleet of self-driving cars, comprising real-world traffic scenarios from various urban environments.

Through the Waymo Motion Prediction Challenge, participants have the opportunity to benchmark their solutions against state-of-the-art methods and showcase their innovations in motion prediction. By evaluating and comparing different approaches, this challenge aims to foster collaboration, accelerate research, and drive advancements in the field.

This paper aims to provide an overview of the Waymo Motion Prediction Challenge, discussing its significance, objectives, and impact on the development of autonomous vehicles. We will explore the challenges associated with accurate motion prediction, the available dataset, evaluation metrics, and some notable approaches employed by participants.

Chapter 2

Literature review

This chapter contains the survey of some papers published on Motion Prediction which I have refereed during my project work.

Waymo motion prediction competition has been a legacy platform for benchmarking best efforts for motion prediction in the self-driving industry and research. Most automated driving systems incorporate data from a variety of sensors, devices, including cameras, radar, LiDAR, wheel odometry, GPS, and IMUs. Sensor signals from various sensors are included in latest datasets for autonomous vehicles. Using 22 sequences of synchronised stereo camera and LiDAR sensor data, the multi-sensor KITTI Dataset, developed by Geiger et al. in 2012, makes it possible to perform tasks like visual navigation system, scene flow estimations, and 3D object detection and tracking.

A self-driving automobile needs to be able to comprehend its connections with other road users in order to reason and carry out the safest motion plan. Modeling such behaviour is not simple and requires many variables, including demographics, traffic volume, the environment, local regulations, contextual clues, etc. The relevant literature is thoroughly reviewed in this article. We specifically identify and categorise motion prediction research for two kinds of road users, namely cars and pedestrians.

Since 2020, the Waymo Open Dataset has been providing the research communities with high-quality data gathered from camera and LiDAR sensors in actual self-driving situations, which have allowed for a lot of new and intriguing studies. By enlarging initial Motion Datasets, the Waymo Open DatasetTsung-Yi Lin & Doll´ar. (2017) Task at CVPR 2022 launches a brand-new challenge called "Occupancy and Flow

Prediction". This new form of representation Anthony Hu & Kendall. (2021) mitigates the shortcomings of the existing representation such as trajectory sets and occupancy. By predicting flow fields All agents in the scene can have their motion and location probabilities simultaneously recorded using occupancy and occupancy together. More specifically, the system must forecast the occupancy and flow fields for all cars in 8 seconds, given a one-second history of the agents in the scene. The evaluation metric is calculated using Flow-Grounded Occupancy, which measures the quality of predicted occupancy and flows jointly. The algorithm that achieves the best AUC score with Flow- Grounded Occupancy wins the challenge.

The field of motion prediction has been extensively studied. One of the simplest (albeit occasionally beneficial) techniques is to employ physics-based models, such as the constant velocity, constant acceleration, and constant yaw rate and velocity models. However, dynamics models are unable to take into consideration the road environment or interactions between agents. As a result, certain techniques, such as the intelligent driver modelM. Treiber & Helbing (2000) and the social forces model, concentrate on mathematically defining these interactions. Although these model-based solutions are simple to use and computationally effective, their performance and accuracy are constrained. Motion prediction has been done using deep neural networks in a significant amount of recent literature, as learning-based techniques have attracted significant interest. To accurately depict the intricacy of road infrastructure, they rely on a vast amount of observational data. Encounters with numerous agents and their behaviors in real-world settings, displaying excellent prediction accuracy as well as versatility.

They used the Velodyne HDL-64E LIDAR in their work. J. & S. (2010) suggested a localization method that uses offline grid maps of the reflectance intensity distribution of the environment measured by a LIDAR scanner (remission grid maps). The Velodyne HDL-64E's laser beams are calibrated using an unsupervised calibration technique so that they react uniformly to objects of the same brightness as detected by the LIDAR. The autonomous vehicle position is calculated using a 2-dimensional histogram filter, Thrun S. & D. (2005). The filter, as usual, consists of two parts: the measurement update (or correction), which boosts confidence in our estimate based on sensor data, and the motion update (or prediction), which estimates the automobile's position based on its motion. In the motion update, the motion of the vehicle is modeled as a random walk with Gaussian noise drifts from a dead reckoning coordinate system to the global coordinate system of the offline map. This computation was done utilizing the inertial update of an Applanix POS LV-420 position and orientation system. The similarity between the online and offline remission maps is used in the measurement stage to account for various displacements. The histogram filter's cells of the histogram correspond to each displacement. They utilize the center of mass of the probability distribution represented by the histogram to condense the histogram into a single pose estimate. However, the authors don't explain how they determine the orientation. The Root Mean Squared (RMS) lateral and longitudinal errors for their methodology were 9 and 12 cm, respectively.

A MCL localization technique that contrasts remission maps with satellite aerial maps was put forward in Veronese L. (2015). Remission maps are created online from LIDAR reflectance intensity data, and aerial maps are obtained offline from web resources like OpenStreetMap. By comparing remission maps to aerial maps and computing the particles likelihood using the Normalised Mutual Information (NMI) metric, the MCL method is used to determine the pose of the automobile. The approach achieved a position estimation accuracy of 0.89 m on a 6.5 km dataset compiled by the self-driving automobile IARA. This approach has the benefit of not requiring the creation of a map especially for it.

A localization technique based on the detection of road features was proposed in A.Y. & D.F. (2015). Their curb detection system analyses the separation between successive concentric measurements (or rings) produced by a multilayer LIDAR (Velodyne HDL-32E scan) using ring compression analysis and least trimmed squares. Otsu thresholdingN. (1979) is used in the road marking detection algorithm to examine LIDAR reflectance intensity data. A grid map stores information on curb and road marking elements. By comparing road features generated from multilayer LIDAR scans to the grid map, a Monte Carlo Localization (MCL) technique is used to estimate the car pose. On the self-driving car CARINA Fernandes L.C. & A. (2014), the approach was evaluated, and results showed lateral and longitudinal localization estimate errors of less than 0.30 m.

A multilayer adaptive Monte Carlo localization (ML-AMCL) technique that works with 3D point registration techniques was proposed in Rohde J. & J.M. (2016). A 2D projection of a 3D point cloud map created using 3D point registration methods is utilised to align the horizontal layers generated from 3D LIDAR readings for calculating the automobile posture. Every pose estimation is subjected to a consistency check against a set of odometry measurements. A final pose estimate is created by merging consistent pose estimates. The technique obtained position estimation errors of 0.25 m in relation to the GPS reference after being tested on actual data. However, because their map is in three dimensions, storage costs are high.

A probabilistic localization method called R.W. & R.M. (2017) was proposed, modelling the world as a multi-resolution map made up of a mixture of Gaussians. The height and reflectance intensity distribution of the surroundings as determined by Velodyne HDL-32E LIDAR scanners are represented in their Gaussian mixture maps. The pose of the automobile is estimated using an Extended Kalman filter (EKF) localization approach by registering 3D point clouds against Gaussian mixture multiresolution maps. The approach had localization estimation errors of roughly 0.15m when it was tested on two autonomous vehicles in bad weather.

In theory, an automated vehicle system can only be termed as an "autonomous" system, when all the dynamic driving tasks, at all driving environments, can be performed by the vehicle's automated system. According to the Federal Automated Vehicles Policy of the US Department of Transportation, a vehicle is denoted as AV if it has levels 3-5 automated systems DoT. (2016). However, these levels of autonomy are not strictly maintained in the literature and any level of autonomy is referred to as autonomous Shladover (2018). Throughout this paper, the term AV will refer to the levels 3-5 automated systems only.

Driving requires a variety of functions, including localization, perception, planning, control, and management Coppola (2016). Information acquisition is a prerequisite to localization, and perception. If all of these functions, including information acquisition, are available in a vehicle, it could definitely be termed as an AV. If any AV has to communicate with other infrastructures to collect information, or to negotiate its maneuvers, it is termed as connected autonomous vehicle (CAV)Shladover (2018), and when any manually driven vehicle, whether manual or automated, has to communicate with other infrastructures to collect information, or to negotiate its maneuvers, it is termed as connected vehicle (CV) Hendrickson (2014)-Coppola (2016). Therefore, CV technology is complimentary or has synergistic effect on the implementation of AV to some extent Shladover (2018), though connectivity is not a mandatory feature of AVs Hendrickson (2014).

The motion prediction problem is initially formulated by researchers as a time series prediction problem, in which a sequence of past states is used to forecast a sequence of future states. In order to forecast motion, RNNs, or more precisely long short-term memory systems, have been widely used. A. Alahi & Savarese (June 2016). Many works use the image structure that rasterizes the driving environment into 2D grids with each pixel representing a semantic class, which can be efficiently processed by CNNs, to include the environment information (for example, road structure, traffic signals), which can be incorporated into motion prediction.

The rasterized images do not, however, explicitly depict the interaction of traffic participants. Since the attention-based feature fusion can effectively represent and model the interaction between agents, many recent efforts aim to integrate graph modelling and the attention mechanism in order to explicitly model the interactions. To account for interactions between several vehicles, for instance,J. Mercat & Gil (2020) uses multi-head self attention, and propose using graph attention networks to extract relational information from the scene graph comprising various agents. We propose a distance attention module to represent the interaction between the target agent and the agents around it because, in a traffic situation, the attention the target agent gives to a surrounding actor is substantially affected by the distance between them.

Although deep learning-based approaches work well, a finer point is still necessary for safety-critical applications. applications like self-driving cars, which are to forecast To make safe selections, there should be a variety of potential future paths, ideally with the likelihood of each occurring. To a solution to this problem, various publications suggest using generative models like conditional variational autoencoderT. Salzmann & Pavone (2020) are examples. However, these techniques can need thousands of samples to find a considerable distribution, which might greatly reduce the rate of inference. To solve this issue, we set up a group of trajectory decoders and use the training strategy described in H. Cui & Djuric (2019) by only training the decoder that is closest to the ground truth trajectory. This may guarantee the variety of the projected trajectories and stabilize training. To forecast a confidence score or a normalized probability for each anticipated trajectory, we additionally add a classification branch to the network.

Motion prediction remains an ongoing challenge in the research community. Open competitions are frequently held encouraging teams to share their best models and iterate on previous approaches. Top-scoring teams often publish their work and share their approaches with the broader community. Several projects that have garnered attention in the community include works by Konev et al., Zhao et al., and Gu et al. Stepan Konev & Sanakoyeu. (2021.)-Hang Zhao (2020.)-Junru Gu & Zhao. (2021.). Though a rich set of approaches exist, including constant velocity and latent behavior approaches, for our purposes we can divide approaches into end-to-end and modular approaches, with the latter having shown significant potential in recent competition settings. End-to-end approaches primarily vary in preprocessing, feature use, and model architecture. One of the primary inspirations for this project is a naïve CNN model that rasterizes the scenario and leverages convolutional layers to generate predictions directly Stepan Konev & Sanakoyeu. (2021.). On the other hand, modular approaches may differ in the intermediate steps taken to perform the overall motion prediction task. For instance, two of the most promising approaches that achieve state-of-the-art performance in recent Waymo motion prediction challenges are the Target-driveN Trajectory (TNT) and the DenseTNT models, both of which first rank plausible endpoints from the map, then construct trajectories conditioned on these endpoints, and finally select the best trajectories Hang Zhao (2020.)-Junru Gu & Zhao. (2021.).

Chapter 3

Problem Statement and objectives

3.1 Problem Statement

Deep learning is a rapidly emerging and predominantly used method for a wide range of applications like Image processing, transportation(Self-driving car), Electronics, and digital media platforms(recommendation systems).

Given agents' tracks for the past 1 second on a corresponding map, predict the positions of up to 8 agents for 8 seconds into the future. To enable the motion prediction challenge, the ground truth future data for the test set is hidden from challenge participants. As such, the test sets contain only 1 second of history data. The validation sets contain the ground truth future data for use in model development. In addition, the test and validation sets provide a list of up to 8 object tracks in the scene to be predicted. These are selected to include interesting behavior and a balance of object types.

The Waymo Motion Prediction Challenge aims to address the complex problem of predicting the future motion of various road agents in a self-driving car environment. Given the rich sensor data collected from Waymo's autonomous vehicles, the challenge participants are required to develop innovative machine-learning models that can accurately anticipate the future trajectories of different road users, such as vehicles, cyclists, and pedestrians.

The challenge involves predicting the future motion of road agents in a highly dynamic and unpredictable urban environment. Participants are provided with a comprehensive dataset containing high-resolution sensor information, including lidar, radar, and camera data, collected from Waymo's autonomous vehicles. The dataset covers a wide range of scenarios, including different weather conditions, traffic patterns, and road layouts.

The main goal of the challenge is to enable self-driving cars to make informed and reliable decisions by accurately anticipating the behavior of other road users. By accurately predicting the future trajectories of vehicles, cyclists, and pedestrians, autonomous vehicles can proactively plan their actions, such as lane changes, stopping, or yielding, to ensure safe and efficient navigation in complex traffic situations.

Participants in the Waymo Motion Prediction Challenge are expected to develop machine learning models that can analyze the available sensor data and generate precise trajectory predictions for various road agents. The models should take into account the inherent uncertainties and dynamics of the real-world environment, including the interactions between different road users and the potential influence of external factors such as traffic signals or pedestrian crossings.

The challenge provides an opportunity to advance the state-of-the-art in motion prediction for autonomous driving and contribute to the development of safer and more reliable self-driving systems. By accurately predicting the future motion of road agents, participants can help enhance the decision-making capabilities of autonomous vehicles and enable them to navigate complex urban environments with increased efficiency and safety.

3.2 Objectives

- **Objective-1:** Collection of a dataset from the Waymo website the data is available in the form of .tfrecord. Then it is converted to .npz format.
- **Objective-2:** The dataset is split into training, validation and test set, corresponding to 70%, 15%, and 15% of the data. Then the pre-processing of data we need to do in this step it includes a "normalization" & "rasterization".
- **Objective-3:** Building the model for the motion prediction tasks, LSTM Goal prediction model is to estimate the endpoint of objects 8 seconds in the future provided with 1 second of past context. In this model test data is applied to find the FDE metric.
- **Objective-4:** Later, the Trajectory completion model receives as input a preprocessed scenario and couples it with the estimated endpoint given from the goal prediction model. The goal of this model is to predict 80 coordinates, ten for each second in the future, for each object conditioned on their goal. In this model test data is applied to find the ADE metric.
- **Objective-5:** Finally, the combination of two models is used to evaluate performance by requiring models to predict trajectories for up to eight objects in a given scenario.
- **Objective-6:** Using the above model we are finding the ADE & FDE metric for motion prediction challenge. Finally, achieving decent performance in goal prediction and high performance in the trajectory completion module.

Chapter 4

Experimental Setup

In this chapter, I have mentioned the prerequisite steps to be followed to perform this experiment. It includes installation of various packages and modules required to build the data preparation for train, test, and validation datasets. These packages are required to execute the model to predict the objects.

4.1 Installation of Specific Packages

!pip install: This command is used in Google Colab to install Python packages. The exclamation mark "!" indicates that it's a shell command executed within the notebook environment.

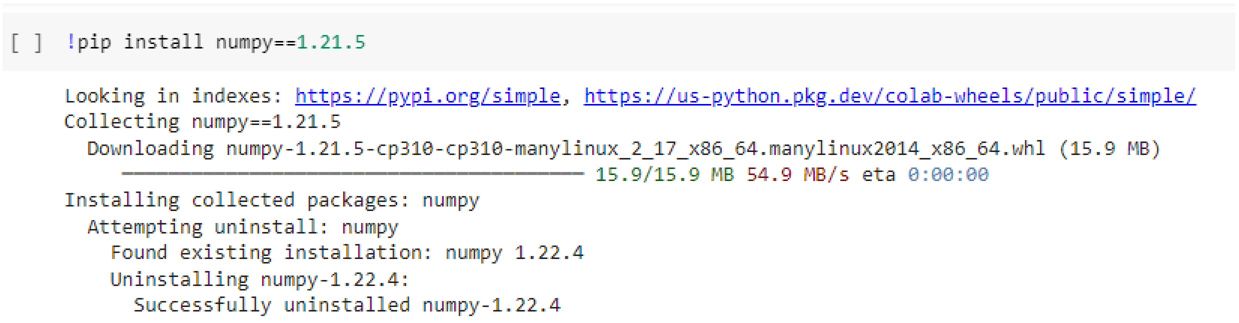
4.1.1 Numpy

- **numpy:** Refers to the name of the package being installed, in this case, NumPy. NumPy is a powerful library for numerical computing in Python, providing support for multi-dimensional arrays, mathematical functions, and linear algebra operations.

- **==1.21.5:** This part specifies the version of NumPy to be installed. By using the double equal sign "==", you're requesting the exact version 1.21.5. It ensures that only this specific version will be installed.

```
!pip install numpy==1.21.5
```

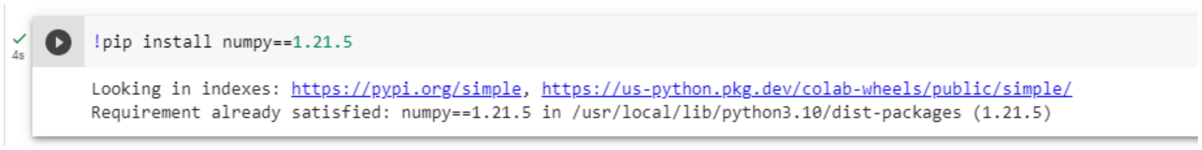
When you run this command in a code cell in Google Colab, it connects to the Python Package Index (PyPI) repository and downloads the specified version of NumPy. Once the installation is complete, you'll be able to use NumPy version 1.21.5 in your Colab notebook for any required numerical computations.



```
[ ] !pip install numpy==1.21.5

    Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
    Collecting numpy==1.21.5
      Downloading numpy-1.21.5-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (15.9 MB)
         ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 15.9/15.9 MB 54.9 MB/s eta 0:00:00
    Installing collected packages: numpy
      Attempting uninstall: numpy
        Found existing installation: numpy 1.22.4
        Uninstalling numpy-1.22.4:
          Successfully uninstalled numpy-1.22.4
```

Figure 4.1: Pip install numpy



```
!pip install numpy==1.21.5

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: numpy==1.21.5 in /usr/local/lib/python3.10/dist-packages (1.21.5)
```

Figure 4.2: successfully installed numpy==1.21.5

NumPy (Numerical Python) is a fundamental package for scientific computing in Python. It provides powerful tools for working with arrays, matrices, and numerical operations. Here are some of the main uses of NumPy in Python:

1. **Multi-dimensional arrays:** NumPy introduces the ndarray object, which allows you to create and manipulate multi-dimensional arrays efficiently. These arrays can have any number of dimensions, and they provide a fast and efficient way to store and manipulate large amounts of data.

2. **Mathematical operations:** NumPy provides a wide range of mathematical functions and operations optimized for arrays. You can perform element-wise operations, vectorized calculations, linear algebra operations, Fourier transforms, and more. NumPy functions are designed to work efficiently with large arrays, making it a preferred choice for numerical computations.

3. **Data manipulation:** NumPy offers various functions for manipulating and transforming array data. You can reshape arrays, transpose dimensions, concatenate arrays, split arrays, and apply logical operations. NumPy also provides

tools for indexing, slicing, and iterating over arrays, allowing you to extract and modify specific parts of the data.

4. **Random number generation:** NumPy includes a random module that enables the generation of random numbers. You can generate random samples from different probability distributions, set seed values for reproducibility, and simulate random processes.

5. **Integration with other libraries:** NumPy serves as the foundation for many other scientific computing libraries in Python. It integrates seamlessly with libraries such as SciPy, Pandas, Matplotlib, and scikit-learn, providing a consistent array-based interface for data manipulation, numerical computing, data visualization, and machine learning tasks.

6. **Performance optimization:** NumPy's underlying implementation is written in highly optimized C code, which makes it significantly faster than performing similar computations using pure Python lists. By leveraging NumPy, you can achieve better performance for numerical computations and take advantage of vectorized operations that operate on entire arrays instead of looping over individual elements.

These are just a few examples of how NumPy is used in Python. Its versatility, efficiency, and extensive ecosystem of libraries make it a fundamental tool for scientific computing, data analysis, machine learning, and various other domains in Python.

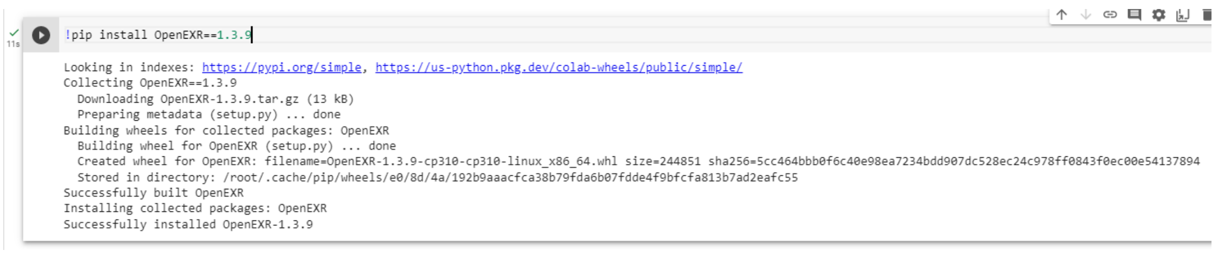
4.1.2 openEXR

- **openexr:** Refers to the name of the package being installed, which is OpenEXR. OpenEXR is a library for high dynamic range (HDR) image file format developed by Industrial Light Magic (ILM).

- ==**1.3.9:** This part specifies the version of OpenEXR to be installed. By using the double equal sign "==", you're requesting the exact version 1.3.9. It ensures that only this specific version will be installed.

```
!pip install OpenEXR==1.3.9
```

When you run this command in a code cell in Google Colab, it connects to the Python Package Index (PyPI) repository and downloads the specified version of OpenEXR. Once the installation is complete, you'll be able to use OpenEXR version 1.3.9 in your Colab notebook for any required HDR image processing tasks.



```
!pip install OpenEXR==1.3.9
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting OpenEXR==1.3.9
  Downloading OpenEXR-1.3.9.tar.gz (13 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: OpenEXR
  Building wheel for OpenEXR (setup.py) ... done
  Created wheel for OpenEXR: filename=OpenEXR-1.3.9-cp310-cp310-linux_x86_64.whl size=244851 sha256=5cc464bbb0f6c40e98ea7234bdd907dc528ec24c978ff0843f0ec00e54137894
  Stored in directory: /root/.cache/pip/wheels/e0/8d/4a/192b9aaacfca38b79fda6b07fdde4f9bfcfa813b7ad2eafc55
Successfully built OpenEXR
Installing collected packages: OpenEXR
Successfully installed OpenEXR-1.3.9
```

Figure 4.3: Pip install openEXR==1.3.9

OpenEXR is a popular file format and library for storing high-dynamic-range (HDR) images and associated data. In Python, the OpenEXR library provides functionality for reading and writing OpenEXR image files, as well as accessing the image data and metadata stored within them. Here are some of the main uses of OpenEXR in Python:

1. **Working with HDR images:** OpenEXR is commonly used for storing HDR images, which contain a wider range of luminance values than standard images. The OpenEXR library allows you to read and write HDR images in Python, preserving the full dynamic range and color accuracy of the image data.

2. **Retaining image fidelity:** OpenEXR uses a lossless compression algorithm to store image data, ensuring that the original image quality is preserved. This is especially important for applications where maintaining high precision and fidelity is crucial, such as visual effects, computer graphics, and scientific visualization.

3. **Accessing image channels and metadata:** OpenEXR files can store multiple image channels, each representing different aspects of the image data (e.g., color channels, alpha channel, depth channel). The OpenEXR library enables you to access individual channels, manipulate their data, and extract relevant information. It also provides access to metadata stored within the file, such as camera settings, exposure information, or custom annotations.

4. **Integration with image processing pipelines:** OpenEXR integrates well with various image processing libraries and frameworks in Python. You can use OpenEXR files as input or output in your image processing pipelines, leveraging the capabilities of other libraries like NumPy, PIL/Pillow, or OpenCV for additional image manipulation and analysis tasks.

5. **Cross-platform compatibility:** OpenEXR is an open standard and is supported on multiple platforms, making it easy to exchange HDR image data between different applications and systems. The OpenEXR library ensures consistent and reliable handling of OpenEXR files across platforms, enabling interoperability in your Python projects.

Whether you're working with HDR image data, need to retain high precision and fidelity, or require access to specific image channels and metadata, the OpenEXR library in Python provides a robust and flexible solution for reading, writing, and manipulating OpenEXR files.

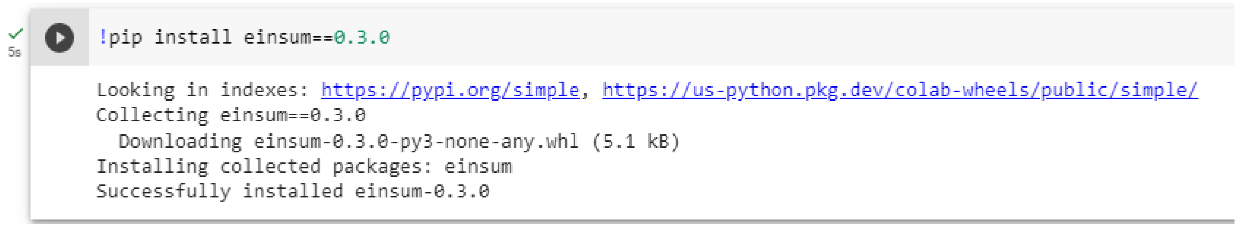
4.1.3 einsum

- **einsum:** Refers to the name of the package being installed, which is einsum. einsum is a library that provides a powerful and concise way to perform Einstein summation notation in NumPy arrays.

- **==0.3.0:** This part specifies the version of einsum to be installed. By using the double equal sign "==", you're requesting the exact version 0.3.0. It ensures that only this specific version will be installed.

```
!pip install einsum==0.3.0
```

When you run this command in a code cell in Google Colab, it connects to the Python Package Index (PyPI) repository and downloads the specified version of einsum. Once the installation is complete, you'll be able to use einsum version 0.3.0 in your Colab notebook for performing Einstein summation operations on NumPy arrays efficiently.



```
!pip install einsum==0.3.0

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting einsum==0.3.0
  Downloading einsum-0.3.0-py3-none-any.whl (5.1 kB)
Installing collected packages: einsum
Successfully installed einsum-0.3.0
```

Figure 4.4: Pip install einsum==0.3.0

The "einsum" function in Python is part of the NumPy library and provides a powerful way to perform Einstein summation notation. Einstein summation is a concise and flexible notation for expressing various operations on multi-dimensional arrays, such as matrix multiplication, dot products, element-wise multiplication, and contraction of indices.

Here are some common uses of "einsum" in Python:

1. **Matrix multiplication:** "einsum"allows you to perform matrix multiplication by specifying the contraction of indices between two arrays. For example, "np.einsum('ij,jk', A, B)" performs matrix multiplication between arrays A and B, summing over the shared index j to produce the resulting matrix.

2. **Dot product:** "einsum" can compute the dot product between two vectors by contracting their indices. For instance, "np.einsum('i,i', u, v)" calculates the dot product between vectors u and v by summing the products of their corresponding elements.

3. **Element-wise multiplication:** "einsum" can be used to perform element-wise multiplication between two arrays. By specifying the same indices for both arrays, einsum will multiply corresponding elements. For example, "np.einsum('i,i', u, v)" computes the element-wise multiplication between vectors u and v.

4. **Index contraction:** "einsum" enables contraction of indices to perform summation or reduction operations. By specifying repeated indices in the notation, you can contract those indices and compute their sum or perform reductions. For instance, "np.einsum('i,i->', u, v)" computes the sum of the element-wise multiplication between vectors u and v.

5. **Advanced operations:** "einsum" supports more complex operations, such

as transpose, broadcasting, and reshaping, by manipulating the indices in the notation. This allows you to express a wide range of linear algebra and tensor operations concisely.

The "einsum" function provides a flexible and efficient way to express and compute various operations on multi-dimensional arrays. It offers a concise notation that can simplify complex calculations and improve code readability. By leveraging the power of "einsum", you can perform a wide range of array operations in a more intuitive and compact manner.

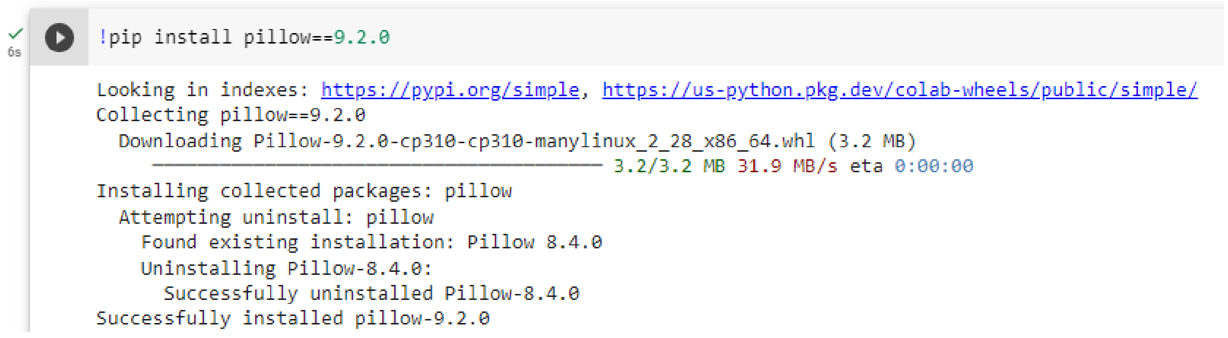
4.1.4 Pillow

- **pillow:** Refers to the name of the package being installed, which is Pillow. Pillow is a Python imaging library that provides a convenient and easy-to-use interface for opening, manipulating, and saving various image file formats.

- **:** This part specifies the version of Pillow to be installed. By using the double equal sign "==", you're requesting the exact version 9.2.0. It ensures that only this specific version will be installed.

```
!pip install pillow==9.2.0
```

When you run this command in a code cell in Google Colab, it connects to the Python Package Index (PyPI) repository and downloads the specified version of Pillow. Once the installation is complete, you'll be able to use Pillow version 9.2.0 in your Colab notebook for performing various image processing tasks, such as resizing, cropping, filtering, and more.



```
!pip install pillow==9.2.0

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pillow==9.2.0
  Downloading Pillow-9.2.0-cp310-cp310-manylinux_2_28_x86_64.whl (3.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.2/3.2 MB 31.9 MB/s eta 0:00:00
Installing collected packages: pillow
  Attempting uninstall: pillow
    Found existing installation: Pillow 8.4.0
    Uninstalling Pillow-8.4.0:
      Successfully uninstalled Pillow-8.4.0
Successfully installed pillow-9.2.0
```

Figure 4.5: Pip install pillow==9.2.0

Pillow is a popular Python library for image processing and manipulation. It provides a wide range of functionalities to work with images, including opening, saving, resizing, cropping, applying filters, and performing various transformations. Here are some common uses of Pillow in Python:

1. **Image loading and saving:** Pillow allows you to open and load different image file formats, such as JPEG, PNG, BMP, TIFF, and GIF. You can also save images in various formats, including converting between formats if needed.

2. **Image resizing and cropping:** Pillow provides methods to resize images to specific dimensions, maintaining aspect ratio if desired. You can also crop images to extract a specific region of interest.

3. **Image manipulation:** Pillow offers numerous image manipulation operations, such as rotating, flipping, and adjusting brightness, contrast, and saturation. You can apply filters, such as blur, sharpen, and edge enhancement, to enhance or modify the appearance of images.

4. **Image enhancement:** Pillow includes functions for enhancing image quality, such as adjusting sharpness, reducing noise, and improving color balance. These operations can help improve the visual appeal and overall quality of images.

5. **Image transformation:** Pillow allows you to perform geometric transformations on images, such as affine transformations (translation, rotation, scaling) and perspective transformations. These transformations can be useful for tasks like image registration, correction, or generating augmented datasets.

6. **Image analysis:** Pillow provides basic functions for analyzing images, such as extracting color histograms, calculating image statistics (mean, median, etc.), and accessing pixel values. These features can be useful for image processing tasks that involve quantitative analysis or feature extraction.

7. **Image rendering and text overlay:** Pillow allows you to draw shapes, lines, and text on images. This feature is useful for annotating images, adding watermarks, or generating custom visualizations.

Pillow is a versatile and widely used library for image processing in Python. Its intuitive API and extensive set of functions make it a valuable tool for various applications, including computer vision, web development, scientific research, and data analysis.

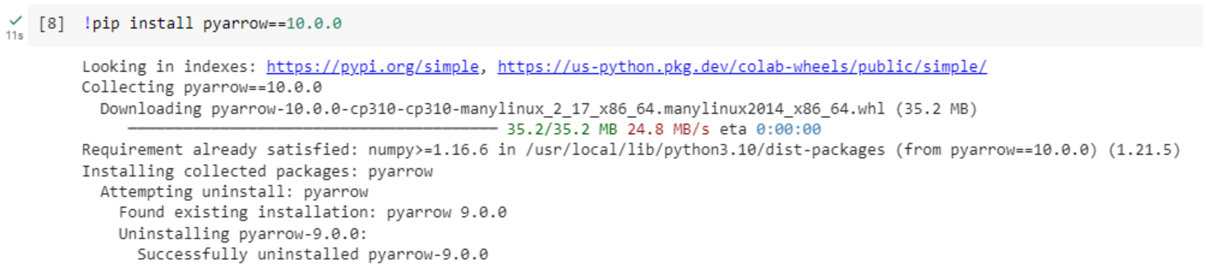
4.1.5 Pyarrow

- **pyarrow:** Refers to the name of the package being installed, which is PyArrow. PyArrow is a Python library that provides bindings to Apache Arrow, a columnar in-memory data format. It enables high-performance data interchange between different systems and languages.

- **==10.0.0:** This part specifies the version of PyArrow to be installed. By using the double equal sign "==", you're requesting the exact version 10.0.0. It ensures that only this specific version will be installed.

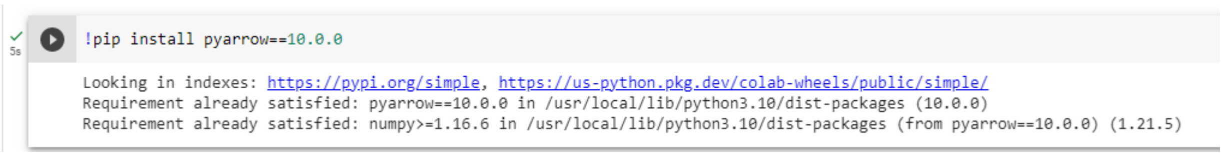
```
!pip install pyarrow==10.0.0
```

When you run this command in a code cell in Google Colab, it connects to the Python Package Index (PyPI) repository and downloads the specified version of PyArrow. Once the installation is complete, you'll be able to use PyArrow version 10.0.0 in your Colab notebook for efficient data manipulation, processing, and interchange across different platforms and languages.



```
✓   [8]  !pip install pyarrow==10.0.0
11s
         Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
         Collecting pyarrow==10.0.0
           Downloading pyarrow-10.0.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (35.2 MB)
              ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 35.2/35.2 MB 24.8 MB/s eta 0:00:00
         Requirement already satisfied: numpy>=1.16.6 in /usr/local/lib/python3.10/dist-packages (from pyarrow==10.0.0) (1.21.5)
         Installing collected packages: pyarrow
           Attempting uninstall: pyarrow
             Found existing installation: pyarrow 9.0.0
             Uninstalling pyarrow-9.0.0:
               Successfully uninstalled pyarrow-9.0.0
```

Figure 4.6: Pip install pyarrow



```
!pip install pyarrow==10.0.0

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pyarrow==10.0.0 in /usr/local/lib/python3.10/dist-packages (10.0.0)
Requirement already satisfied: numpy>=1.16.6 in /usr/local/lib/python3.10/dist-packages (from pyarrow==10.0.0) (1.21.5)
```

Figure 4.7: successfully installed pyarrow==10.0.0

PyArrow is a powerful Python library that provides tools and functionalities for working with columnar data and enables efficient interchange of data between different systems and programming languages. Here are some common uses of PyArrow in Python:

1. **Columnar data manipulation:** PyArrow provides an in-memory columnar data structure called "Table," which allows you to efficiently store, manipulate, and query large datasets. You can perform various operations on columns, such as filtering, aggregating, sorting, and joining, using a concise and efficient syntax.

2. **Interoperability:** PyArrow facilitates data interchange between different systems and programming languages. It allows you to convert data between different formats, such as pandas DataFrames, NumPy arrays, and Apache Arrow format. This enables seamless integration with other data processing and analysis tools.

3. **High-performance analytics:** PyArrow leverages the capabilities of Apache Arrow, a columnar in-memory data format, to enable fast and efficient data processing. It provides functions for performing vectorized operations on large datasets, leveraging CPU parallelism for improved performance.

4. **Data serialization and deserialization:** PyArrow offers serialization and deserialization capabilities to efficiently store and transport data. It provides methods to serialize complex data structures, such as nested arrays, and enables efficient data transfer between different processes or across network boundaries.

5. **Integration with other libraries:** PyArrow integrates well with other popular Python libraries, such as pandas, NumPy, and scikit-learn. It allows you to seamlessly convert data between these libraries and PyArrow's columnar format, enabling efficient data processing and analysis workflows.

6. **Big data and distributed computing:** PyArrow can be used in conjunction with distributed computing frameworks, such as Apache Spark or Dask, to enable efficient data processing on large-scale distributed systems. It provides tools for converting data between distributed formats and performing distributed computations using columnar data structures.

Overall, PyArrow is a versatile library that enables efficient data manipulation, interchange, and analysis in Python. It is particularly useful for working with large datasets, performing high-performance analytics, and integrating with other data processing tools and frameworks.

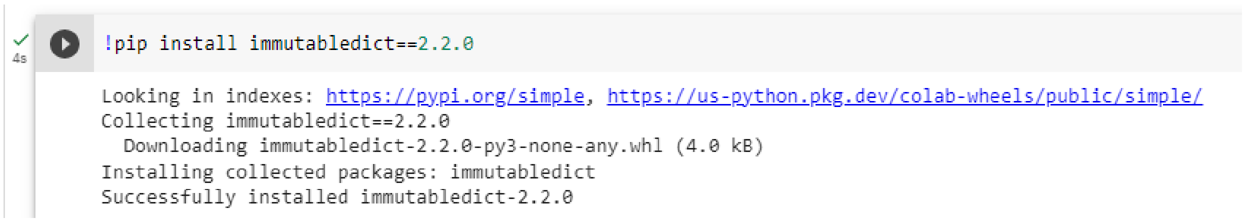
4.1.6 Immutabledict

- **immutabledict:** Refers to the name of the package being installed, which is immutabledict. Immutabledict is a Python library that provides an immutable dictionary implementation. It allows you to create dictionaries that cannot be modified once created, providing an additional level of safety and guaranteeing that the data within the dictionary remains unchanged.

- **==2.2.0:** This part specifies the version of immutabledict to be installed. By using the double equal sign "==", you're requesting the exact version 2.2.0. It ensures that only this specific version will be installed.

```
!pip install immutabledict==2.2.0
```

When you run this command in a code cell in Google Colab, it connects to the Python Package Index (PyPI) repository and downloads the specified version of immutabledict. Once the installation is complete, you'll be able to use immutabledict version 2.2.0 in your Colab notebook to create immutable dictionaries and work with them in your Python code.



```
!pip install immutabledict==2.2.0

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting immutabledict==2.2.0
  Downloading immutabledict-2.2.0-py3-none-any.whl (4.0 kB)
Installing collected packages: immutabledict
Successfully installed immutabledict-2.2.0
```

Figure 4.8: pip install immutabledict==2.2.0

The "immutabledict" is a Python library that provides an immutable dictionary data structure. It is similar to the built-in "dict" in Python but with the key distinction that once created, an "immutabledict" cannot be modified. Here are some common use cases and benefits of using "immutabledict" in Python:

1. **Immutable data structures:** In certain scenarios, you may need to ensure that a dictionary's contents remain unchanged. Immutable dictionaries are useful when you want to prevent accidental modifications or need a hashable object for use as a key in another dictionary.

2. **Hashability and key usage:** Immutable dictionaries, like immutabledict, can be used as keys in other dictionaries or elements in sets since they guarantee immutability. This can be helpful when you need to create composite keys or maintain a set of unique dictionaries.

3. **Caching and memoization:** Immutable dictionaries are often used in caching and memoization scenarios, where the function's results are cached using the function's arguments as keys. Since "immutabledict" objects are immutable and hashable, they can be used as keys in a cache dictionary.

4. **Functional programming:** Immutable data structures align with functional programming principles, where immutability helps ensure referential transparency and makes code easier to reason about. Immutable dictionaries can be used in functional programming paradigms for managing state and performing transformations.

5. **Thread safety:** Immutable dictionaries are inherently thread-safe since they cannot be modified once created. This makes them suitable for concurrent programming scenarios where multiple threads or processes access shared data structures.

6. **Performance optimizations:** Immutable dictionaries can offer performance benefits in specific situations. Since they are immutable, they can be shared across multiple contexts without the need for copying. This can reduce memory usage and improve efficiency in certain use cases.

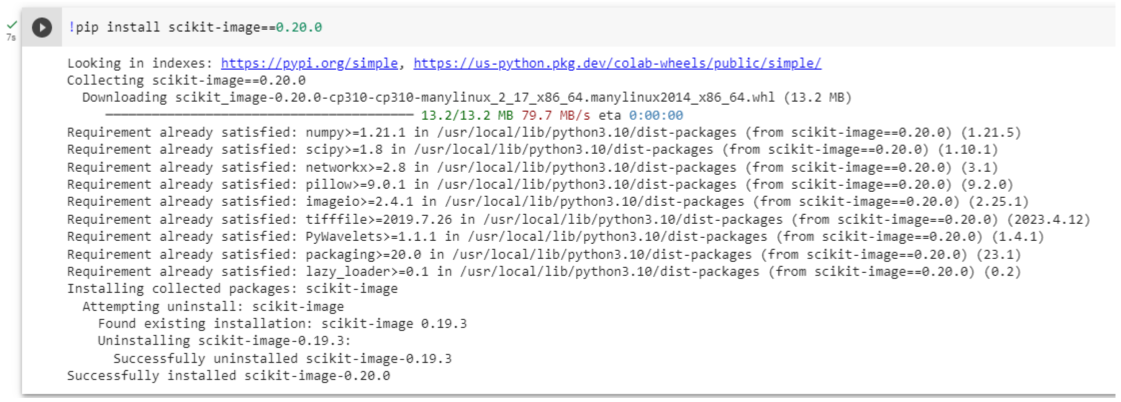
It's important to note that "immutabledict" is not a built-in Python data structure but a separate library that provides the functionality of immutable dictionaries. If you require immutability and the benefits mentioned above, you can import and use the "immutabledict" library in your Python code.

4.1.7 Scikit-image

- **scikit-image:**Refers to the name of the package being installed, which is scikit-image. scikit-image is a Python library for image processing. It provides a collection of algorithms and utilities to manipulate, analyze, and enhance digital images.
- **==0.20.0:** This part specifies the version of scikit-image to be installed. By using the double equal sign "==", you're requesting the exact version 0.20.0. It ensures that only this specific version will be installed.

```
!pip install scikit-image==0.20.0
```

When you run this command in a code cell in Google Colab, it connects to the Python Package Index (PyPI) repository and downloads the specified version of scikit-image. Once the installation is complete, you'll be able to use scikit-image version 0.20.0 in your Colab notebook for various image processing tasks, such as filtering, segmentation, feature extraction, and more. It provides a powerful and convenient API to work with images in Python.



```
!pip install scikit-image==0.20.0

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting scikit-image==0.20.0
  Downloading scikit_image-0.20.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (13.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.2/13.2 MB 79.7 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image==0.20.0) (1.21.5)
Requirement already satisfied: scipy>=1.8 in /usr/local/lib/python3.10/dist-packages (from scikit-image==0.20.0) (1.10.1)
Requirement already satisfied: networkx>=2.8 in /usr/local/lib/python3.10/dist-packages (from scikit-image==0.20.0) (3.1)
Requirement already satisfied: pillow>=9.0.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image==0.20.0) (9.2.0)
Requirement already satisfied: imageio>=2.4.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image==0.20.0) (2.25.1)
Requirement already satisfied: tifffile>=2019.7.26 in /usr/local/lib/python3.10/dist-packages (from scikit-image==0.20.0) (2023.4.12)
Requirement already satisfied: PyWavelets>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image==0.20.0) (1.4.1)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from scikit-image==0.20.0) (23.1)
Requirement already satisfied: lazy_loader>=0.1 in /usr/local/lib/python3.10/dist-packages (from scikit-image==0.20.0) (0.2)
Installing collected packages: scikit-image
  Attempting uninstall: scikit-image
    Found existing installation: scikit-image 0.19.3
    Uninstalling scikit-image-0.19.3:
      Successfully uninstalled scikit-image-0.19.3
Successfully installed scikit-image-0.20.0
```

Figure 4.9: pip install scikit-image==0.20.0

Scikit-image is a Python library that is widely used for image processing and computer vision tasks. It provides a comprehensive set of functions and algorithms for various image-related operations. Here are some common uses of scikit-image in Python:

1. **Image preprocessing:** Scikit-image offers a range of preprocessing techniques for cleaning and enhancing images. It provides functions for tasks like denoising, smoothing, histogram equalization, and contrast adjustment, which can help improve image quality and prepare images for further analysis.

2. **Feature extraction:** Scikit-image includes methods for extracting various features from images, such as edges, corners, contours, and blobs. These features can be used for object detection, image segmentation, and pattern recognition tasks.

3. **Image segmentation:** Scikit-image provides algorithms for segmenting images into meaningful regions. It includes techniques like thresholding, region growing, and watershed segmentation, which can help separate objects or regions of interest from the background.

4. **Morphological operations:** Scikit-image offers morphological operations, such as dilation, erosion, opening, and closing, for manipulating binary images and extracting image components based on shape and size criteria.

5. **Image filtering:** Scikit-image includes a variety of image filtering functions, including Gaussian smoothing, median filtering, and edge-preserving filters. These

filters can be applied to remove noise, blur images, enhance edges, or perform other specific filtering tasks.

6. **Image registration:** Scikit-image provides methods for image registration, which involves aligning multiple images based on their content. This can be useful for tasks like image stitching, super-resolution, and image alignment in medical imaging.

7. **Image analysis and measurements:** Scikit-image offers tools for analyzing and measuring properties of image regions or objects. It provides functions to calculate properties like area, perimeter, intensity, moments, and texture features, which can be used for quantitative analysis and object characterization.

8. **Visualization:** Scikit-image provides functions for visualizing images and their associated features. It includes tools for displaying images, drawing overlays, plotting histograms, and visualizing image transformations.

Scikit-image is a widely used and well-documented library in the field of image processing. It integrates well with other Python libraries, such as NumPy and matplotlib, and provides an extensive collection of functions and algorithms for working with images. Whether you're working on computer vision tasks, image analysis, or image preprocessing, scikit-image offers a powerful toolkit to help you accomplish your goals.

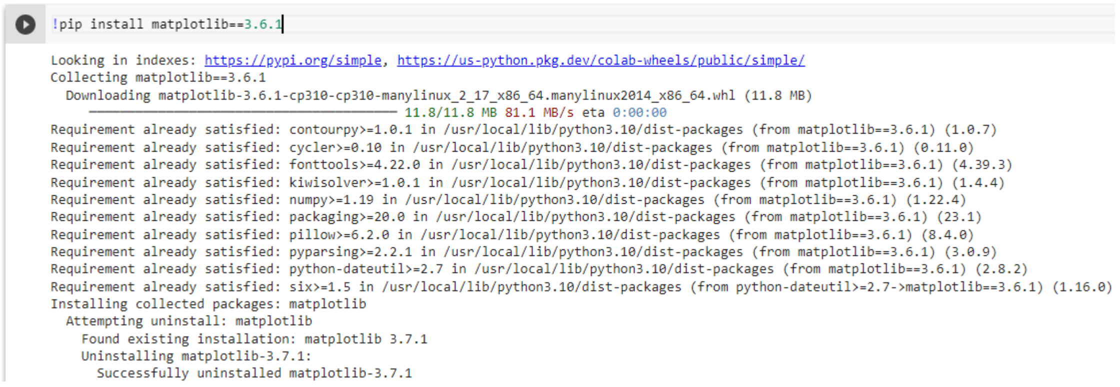
4.1.8 Matplotlib

- **matplotlib:** Refers to the name of the package being installed, which is matplotlib. matplotlib is a Python library for creating visualizations and plots. It provides a wide range of functionalities for creating line plots, bar plots, scatter plots, histograms, and more.

- ==**3.6.1:** This part specifies the version of matplotlib to be installed. By using the double equal sign "==", you're requesting the exact version 3.6.1. It ensures that only this specific version will be installed.

```
!pip install matplotlib==3.6.1
```

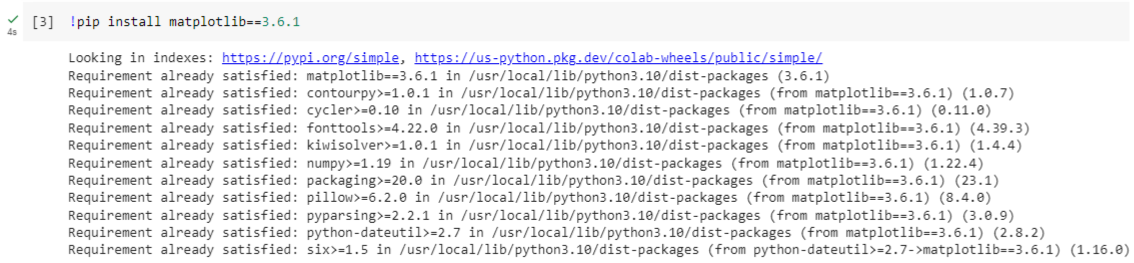
When you run this command in a code cell in Google Colab, it connects to the Python Package Index (PyPI) repository and downloads the specified version of matplotlib. Once the installation is complete, you'll be able to use matplotlib version 3.6.1 in your Colab notebook for creating visualizations and plots. It provides a wide range of functionalities for generating high-quality, publication-ready figures and charts. With matplotlib, you can create various types of plots, including line plots, scatter plots, bar plots, histograms, pie charts, 3D plots, and many more.



```
!pip install matplotlib==3.6.1

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting matplotlib==3.6.1
  Downloading matplotlib-3.6.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.8/11.8 MB 81.1 MB/s eta 0:00:00
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.6.1) (1.0.7)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.6.1) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.6.1) (4.39.3)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.6.1) (1.4.4)
Requirement already satisfied: numpy>=1.19 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.6.1) (1.22.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.6.1) (23.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.6.1) (8.4.0)
Requirement already satisfied: pyparsing>=2.2.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.6.1) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.6.1) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib==3.6.1) (1.16.0)
Installing collected packages: matplotlib
  Attempting uninstall: matplotlib
    Found existing installation: matplotlib 3.7.1
    Uninstalling matplotlib-3.7.1:
      Successfully uninstalled matplotlib-3.7.1
```

Figure 4.10: Pip install matplotlib



```
[3] !pip install matplotlib==3.6.1

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: matplotlib==3.6.1 in /usr/local/lib/python3.10/dist-packages (3.6.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.6.1) (1.0.7)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.6.1) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.6.1) (4.39.3)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.6.1) (1.4.4)
Requirement already satisfied: numpy>=1.19 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.6.1) (1.22.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.6.1) (23.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.6.1) (8.4.0)
Requirement already satisfied: pyparsing>=2.2.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.6.1) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.6.1) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib==3.6.1) (1.16.0)
```

Figure 4.11: successfully installed matplotlib==3.6.1

Matplotlib is a popular Python library for creating static, animated, and interactive visualizations. It provides a wide range of functions and tools for generating various types of plots and charts. Here are some common uses of Matplotlib in Python:

1. **Line plots:** Matplotlib can be used to create line plots, which are useful for visualizing trends and relationships between variables over continuous or categorical data.

2. **Scatter plots:** Scatter plots are effective for visualizing the distribution and relationship between two variables. Matplotlib allows you to create scatter plots

with different markers, colors, and sizes to represent additional dimensions of the data.

3. **Bar plots:** Bar plots are commonly used for displaying and comparing categorical data. Matplotlib provides functions to create vertical, horizontal, stacked, and grouped bar plots, which are helpful for visualizing data across different categories or groups.

4. **Histograms:** Matplotlib enables you to generate histograms to visualize the distribution of numerical data. Histograms divide the data into bins and display the frequency or density of data points within each bin.

5. **Pie charts:** Matplotlib allows you to create pie charts to represent data proportions or percentages. Pie charts are useful for visualizing categorical data and displaying the contribution of each category to the whole.

6. **Box plots:** Box plots, also known as whisker plots, provide a visual summary of the distribution of numerical data. Matplotlib allows you to create box plots to visualize the median, quartiles, and outliers of a dataset.

7. **Heatmaps:** Matplotlib enables the creation of heatmaps, which use color intensity to represent the values of a matrix or a 2D array. Heatmaps are commonly used in fields such as data analysis, image processing, and correlation analysis.

8. **3D plots:** Matplotlib supports the creation of 3D plots for visualizing data in three dimensions. It includes functions for generating surface plots, scatter plots, and wireframe plots to represent complex datasets.

9. **Customization:** Matplotlib provides extensive customization options to modify plot properties, including colors, markers, linestyles, labels, legends, and annotations. You can control axis limits, ticks, and labels, add titles and subtitles, and adjust the overall layout of the plot.

10. **Interactive visualizations:** Matplotlib can be used in combination with other libraries, such as Jupyter Notebook or IPython, to create interactive plots. This allows users to explore and manipulate the plots dynamically, enhancing the data analysis experience.

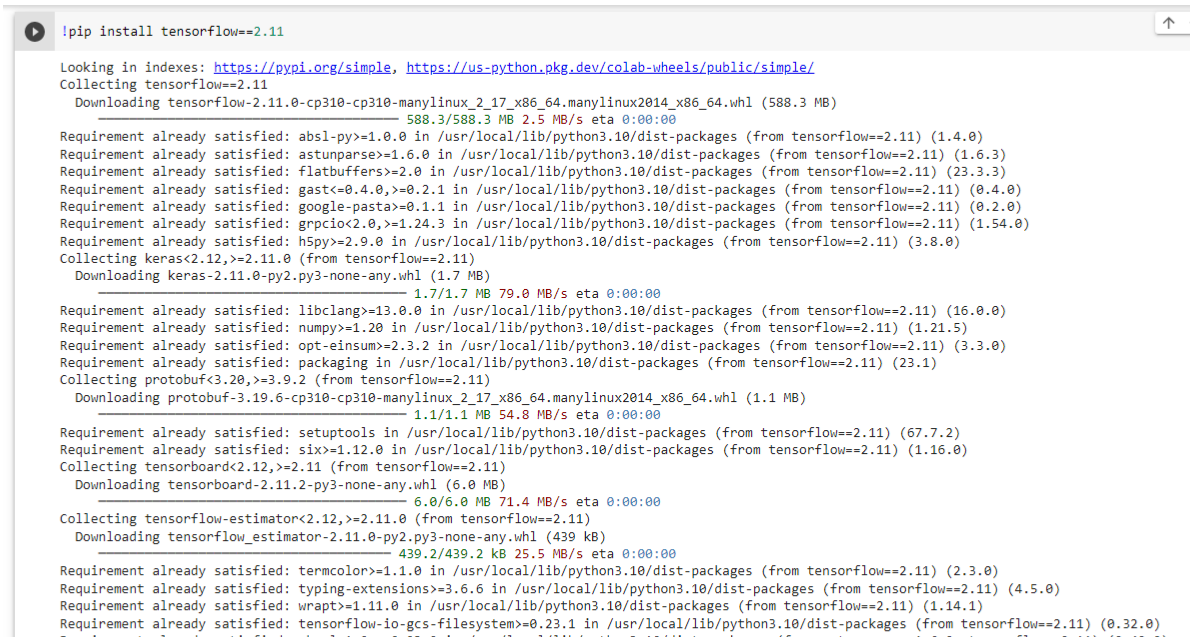
Matplotlib is a versatile and widely-used library for data visualization in Python. Its extensive functionality, flexibility, and customization options make it suitable for a wide range of applications, including scientific research, data analysis, machine learning, and data exploration.

4.1.9 Tensor Flow

- **tensorflow:**Refers to the name of the package being installed, which is tensorflow. TensorFlow 2.11 will be the last TF version to support Python 3.7.// tensorflow is a Python library for image processing. It provides a collection of algorithms and utilities to manipulate, analyze, and enhance digital images.
- ==**0.20.0:** This part specifies the version of tensorflow to be installed. By using the double equal sign "==", you're requesting the exact version 2.11. It ensures that only this specific version will be installed.

```
!pip install tensorflow==2.11
```

When you run this command in a code cell in Google Colab, it connects to the Python Package Index (PyPI) repository and downloads the specified version of tensorflow. Once the installation is complete, you'll be able to use tensorflow version 2.11 in your Colab notebook.



```
!pip install tensorflow==2.11

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting tensorflow==2.11
  Downloading tensorflow-2.11.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (588.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 588.3/588.3 MB 2.5 MB/s eta 0:00:00
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.11) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.11) (1.6.3)
Requirement already satisfied: flatbuffers>=2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.11) (23.3.3)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.11) (0.4.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.11) (0.2.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.11) (1.54.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.11) (3.8.0)
Collecting keras<2.12,>=2.11.0 (from tensorflow==2.11)
  Downloading keras-2.11.0-py2.py3-none-any.whl (1.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 79.0 MB/s eta 0:00:00
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.11) (16.0.0)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.11) (1.21.5)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.11) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.11) (23.1)
Collecting protobuf<3.20,>=3.9.2 (from tensorflow==2.11)
  Downloading protobuf-3.19.6-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 54.8 MB/s eta 0:00:00
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.11) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.11) (1.16.0)
Collecting tensorboard<2.12,>=2.11 (from tensorflow==2.11)
  Downloading tensorboard-2.11.2-py3-none-any.whl (6.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.0/6.0 MB 71.4 MB/s eta 0:00:00
Collecting tensorflow-estimator<2.12,>=2.11.0 (from tensorflow==2.11)
  Downloading tensorflow_estimator-2.11.0-py2.py3-none-any.whl (439 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 439.2/439.2 kB 25.5 MB/s eta 0:00:00
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.11) (2.3.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.11) (4.5.0)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.11) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.11) (0.32.0)
```

Figure 4.12: Pip install tensorflow==2.11

Tensorflow is a Python library for machine learning and deep learning. It provides a flexible framework for building and training various types of machine learning models, particularly deep neural networks. TensorFlow allows you to efficiently define and execute computational graphs, which represent mathematical operations and transformations on multi-dimensional arrays called tensors.

With TensorFlow, you can perform a wide range of tasks related to machine learning, including:

1. **Building and training neural networks:** TensorFlow provides a high-level API called Keras, which simplifies the process of designing and training deep learning models. It also offers a lower-level API that gives you more control and flexibility in defining your models.

2. **Handling and processing large datasets:** TensorFlow provides efficient data handling mechanisms for loading, preprocessing, and batching datasets. It integrates well with other Python libraries like NumPy and pandas for data manipulation and exploration.

3. **Deploying machine learning models:** TensorFlow enables you to export trained models and deploy them in various environments, including cloud plat-

forms, mobile devices, and embedded systems. It supports model conversion and optimization techniques to make the deployment process more efficient.

4. **Performing distributed computing:** TensorFlow allows you to distribute the training and inference processes across multiple devices and machines, including GPUs and TPUs. This makes it possible to scale your machine learning tasks and leverage the power of parallel computing.

Overall, TensorFlow is a powerful and widely used library in the field of machine learning. Its versatility, performance, and extensive community support make it suitable for both research and production-level machine learning projects.

4.1.10 Waymo open dataset

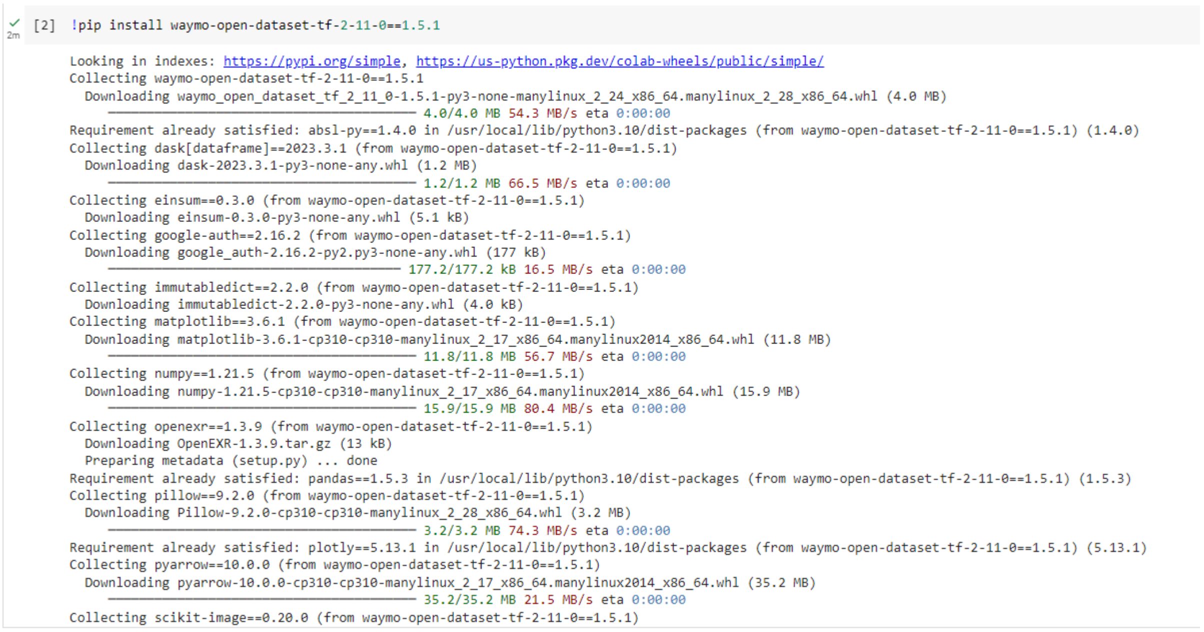
```
!pip install waymo-open-dataset-tf-2-11-0==1.5.1
```

The above command installs a specific version (1.5.1) of the "waymo-open-dataset-tf-2-11-0" package for TensorFlow 2.11. This package is designed specifically for working with the Waymo Open Dataset, a large-scale autonomous driving dataset released by Waymo.

The Waymo Open Dataset contains high-resolution sensor data collected by Waymo self-driving cars. It includes data from various sensors such as lidar, radar, and cameras, along with labeled objects and metadata. The dataset is intended for research and development purposes in the field of autonomous driving, including tasks such as object detection, tracking, and motion prediction.

The "waymo-open-dataset-tf-2-11-0" package provides a set of tools and utilities to work with the Waymo Open Dataset using TensorFlow 2.11. It includes data loading and preprocessing functions, visualization tools, and model evaluation metrics specifically tailored for the Waymo Open Dataset.

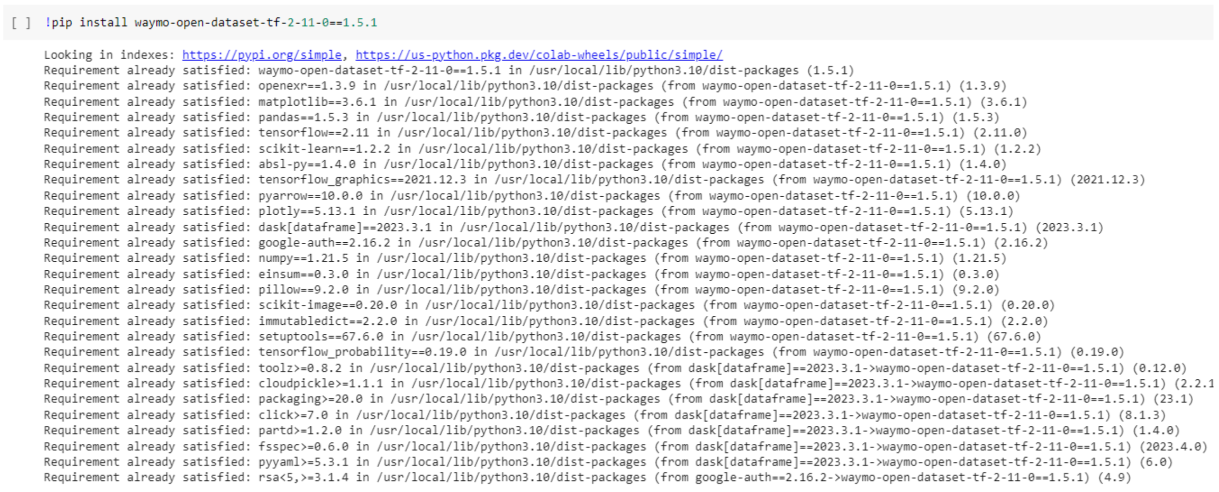
By installing this package and its specified version, you gain access to the functionalities and tools provided by Waymo for working with their dataset. You can use it to load the dataset, preprocess the data, train machine learning models, evaluate performance, and conduct research or development related to autonomous driving systems.



```
[2]  !pip install waymo-open-dataset-tf-2-11-0==1.5.1

     Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
     Collecting waymo-open-dataset-tf-2-11-0==1.5.1
       Downloading waymo_open_dataset_tf_2_11_0-1.5.1-py3-none-manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl (4.0 MB)
          ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.0/4.0 MB 54.3 MB/s eta 0:00:00
     Requirement already satisfied: absl-py==1.4.0 in /usr/local/lib/python3.10/dist-packages (from waymo-open-dataset-tf-2-11-0==1.5.1) (1.4.0)
     Collecting dask[dataframe]==2023.3.1 (from waymo-open-dataset-tf-2-11-0==1.5.1)
       Downloading dask-2023.3.1-py3-none-any.whl (1.2 MB)
          ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.2/1.2 MB 66.5 MB/s eta 0:00:00
     Collecting einsum==0.3.0 (from waymo-open-dataset-tf-2-11-0==1.5.1)
       Downloading einsum-0.3.0-py3-none-any.whl (5.1 kB)
     Collecting google-auth==2.16.2 (from waymo-open-dataset-tf-2-11-0==1.5.1)
       Downloading google_auth-2.16.2-py2.py3-none-any.whl (177 kB)
          ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 177.2/177.2 kB 16.5 MB/s eta 0:00:00
     Collecting immutabledict==2.2.0 (from waymo-open-dataset-tf-2-11-0==1.5.1)
       Downloading immutabledict-2.2.0-py3-none-any.whl (4.0 kB)
     Collecting matplotlib==3.6.1 (from waymo-open-dataset-tf-2-11-0==1.5.1)
       Downloading matplotlib-3.6.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.8 MB)
          ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.8/11.8 MB 56.7 MB/s eta 0:00:00
     Collecting numpy==1.21.5 (from waymo-open-dataset-tf-2-11-0==1.5.1)
       Downloading numpy-1.21.5-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (15.9 MB)
          ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 15.9/15.9 MB 80.4 MB/s eta 0:00:00
     Collecting openexr==1.3.9 (from waymo-open-dataset-tf-2-11-0==1.5.1)
       Downloading OpenEXR-1.3.9.tar.gz (13 kB)
       Preparing metadata (setup.py) ... done
     Requirement already satisfied: pandas==1.5.3 in /usr/local/lib/python3.10/dist-packages (from waymo-open-dataset-tf-2-11-0==1.5.1) (1.5.3)
     Collecting pillow==9.2.0 (from waymo-open-dataset-tf-2-11-0==1.5.1)
       Downloading Pillow-9.2.0-cp310-cp310-manylinux_2_28_x86_64.whl (3.2 MB)
          ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.2/3.2 MB 74.3 MB/s eta 0:00:00
     Requirement already satisfied: plotly==5.13.1 in /usr/local/lib/python3.10/dist-packages (from waymo-open-dataset-tf-2-11-0==1.5.1) (5.13.1)
     Collecting pyarrow==10.0.0 (from waymo-open-dataset-tf-2-11-0==1.5.1)
       Downloading pyarrow-10.0.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (35.2 MB)
          ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 35.2/35.2 MB 21.5 MB/s eta 0:00:00
     Collecting scikit-image==0.20.0 (from waymo-open-dataset-tf-2-11-0==1.5.1)
```

Figure 4.13: Pip install waymo-open-dataset-tf



```
[ ]  !pip install waymo-open-dataset-tf-2-11-0==1.5.1

     Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
     Requirement already satisfied: waymo-open-dataset-tf-2-11-0==1.5.1 in /usr/local/lib/python3.10/dist-packages (1.5.1)
     Requirement already satisfied: openexr==1.3.9 in /usr/local/lib/python3.10/dist-packages (from waymo-open-dataset-tf-2-11-0==1.5.1) (1.3.9)
     Requirement already satisfied: matplotlib==3.6.1 in /usr/local/lib/python3.10/dist-packages (from waymo-open-dataset-tf-2-11-0==1.5.1) (3.6.1)
     Requirement already satisfied: pandas==1.5.3 in /usr/local/lib/python3.10/dist-packages (from waymo-open-dataset-tf-2-11-0==1.5.1) (1.5.3)
     Requirement already satisfied: tensorflow==2.11 in /usr/local/lib/python3.10/dist-packages (from waymo-open-dataset-tf-2-11-0==1.5.1) (2.11.0)
     Requirement already satisfied: scikit-learn==1.2.2 in /usr/local/lib/python3.10/dist-packages (from waymo-open-dataset-tf-2-11-0==1.5.1) (1.2.2)
     Requirement already satisfied: absl-py==1.4.0 in /usr/local/lib/python3.10/dist-packages (from waymo-open-dataset-tf-2-11-0==1.5.1) (1.4.0)
     Requirement already satisfied: tensorflow_graphics==2021.12.3 in /usr/local/lib/python3.10/dist-packages (from waymo-open-dataset-tf-2-11-0==1.5.1) (2021.12.3)
     Requirement already satisfied: pyarrow==10.0.0 in /usr/local/lib/python3.10/dist-packages (from waymo-open-dataset-tf-2-11-0==1.5.1) (10.0.0)
     Requirement already satisfied: plotly==5.13.1 in /usr/local/lib/python3.10/dist-packages (from waymo-open-dataset-tf-2-11-0==1.5.1) (5.13.1)
     Requirement already satisfied: dask[dataframe]==2023.3.1 in /usr/local/lib/python3.10/dist-packages (from waymo-open-dataset-tf-2-11-0==1.5.1) (2023.3.1)
     Requirement already satisfied: google-auth==2.16.2 in /usr/local/lib/python3.10/dist-packages (from waymo-open-dataset-tf-2-11-0==1.5.1) (2.16.2)
     Requirement already satisfied: numpy==1.21.5 in /usr/local/lib/python3.10/dist-packages (from waymo-open-dataset-tf-2-11-0==1.5.1) (1.21.5)
     Requirement already satisfied: einsum==0.3.0 in /usr/local/lib/python3.10/dist-packages (from waymo-open-dataset-tf-2-11-0==1.5.1) (0.3.0)
     Requirement already satisfied: pillow==9.2.0 in /usr/local/lib/python3.10/dist-packages (from waymo-open-dataset-tf-2-11-0==1.5.1) (9.2.0)
     Requirement already satisfied: scikit-image==0.20.0 in /usr/local/lib/python3.10/dist-packages (from waymo-open-dataset-tf-2-11-0==1.5.1) (0.20.0)
     Requirement already satisfied: immutabledict==2.2.0 in /usr/local/lib/python3.10/dist-packages (from waymo-open-dataset-tf-2-11-0==1.5.1) (2.2.0)
     Requirement already satisfied: setuptools==67.6.0 in /usr/local/lib/python3.10/dist-packages (from waymo-open-dataset-tf-2-11-0==1.5.1) (67.6.0)
     Requirement already satisfied: tensorflow_probability==0.19.0 in /usr/local/lib/python3.10/dist-packages (from waymo-open-dataset-tf-2-11-0==1.5.1) (0.19.0)
     Requirement already satisfied: toolz>=0.8.2 in /usr/local/lib/python3.10/dist-packages (from dask[dataframe]==2023.3.1->waymo-open-dataset-tf-2-11-0==1.5.1) (0.12.0)
     Requirement already satisfied: cloudpickle>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from dask[dataframe]==2023.3.1->waymo-open-dataset-tf-2-11-0==1.5.1) (2.2.1
     Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from dask[dataframe]==2023.3.1->waymo-open-dataset-tf-2-11-0==1.5.1) (23.1)
     Requirement already satisfied: click>=7.0 in /usr/local/lib/python3.10/dist-packages (from dask[dataframe]==2023.3.1->waymo-open-dataset-tf-2-11-0==1.5.1) (8.1.3)
     Requirement already satisfied: partd>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from dask[dataframe]==2023.3.1->waymo-open-dataset-tf-2-11-0==1.5.1) (1.4.0)
     Requirement already satisfied: fsspec>=0.6.0 in /usr/local/lib/python3.10/dist-packages (from dask[dataframe]==2023.3.1->waymo-open-dataset-tf-2-11-0==1.5.1) (2023.4.0)
     Requirement already satisfied: pyyaml>=5.3.1 in /usr/local/lib/python3.10/dist-packages (from dask[dataframe]==2023.3.1->waymo-open-dataset-tf-2-11-0==1.5.1) (6.0)
     Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth==2.16.2->waymo-open-dataset-tf-2-11-0==1.5.1) (4.9)
```

Figure 4.14: successfully installed waymo-open-dataset-tf-2-11-0==1.5.1

Chapter 5

Implementation:

In this chapter, I have shown the implementation part of Motion Prediction using goal prediction and trajectory prediction. Therefore, both our goal and trajectory modules generate outputs for eight different objects at a time in a given scenario. This chapter will contain 2 Modules:

- GoalPrediction Model which creates a neural network model for predicting the goals of agents in a scenario.
- Trajectory Prediction consists of two parts one is encoder architecture and the other one is decoder architecture.
- Combined these two models are used to generate outputs for eight different objects at a time in a given scenario and also used to find the metrics of ADE & FDE.

5.1 Dataset and Features

Various datasets have previously been curated and used as standards for motion prediction models, including the Lyft Level 5, NuScenes, Argoverse, and Interactions datasets [*Lyft. Lyft Level 5* (2022), Holger Caesar & Beijbom. (2020), Ming-Fang Chang (2019), Wei Zhan (2019)]. We used the Waymo Open Dataset, a newer dataset that prioritizes modeling 'critical situations', such as scenes at busy street interactions with many pedestrians or scenes with complex acceleration and braking behaviors [*Waymo. Waymo Open Dataset.* (2022), Scott Ettinger (2021)]. The dataset consists of 103,354 segments, each containing 20 seconds of object detection sampled at 10 Hz. The segments are divided into 9-second long segments with a 5-second overlap, which creates a total of 310,062 segments. Each segment has a corresponding scene containing information about the surroundings such as the roads, lanes, and sidewalks, object states and traffic light states. Furthermore, each segment contains pertinent information on up to 128 different objects in the scenario, including their type, size, position, velocity, yaw, among other features. Partially observed objects are marked as such in the scenario data. One example of one scene at two different time points can be seen in figure 5.1 & 5.2.

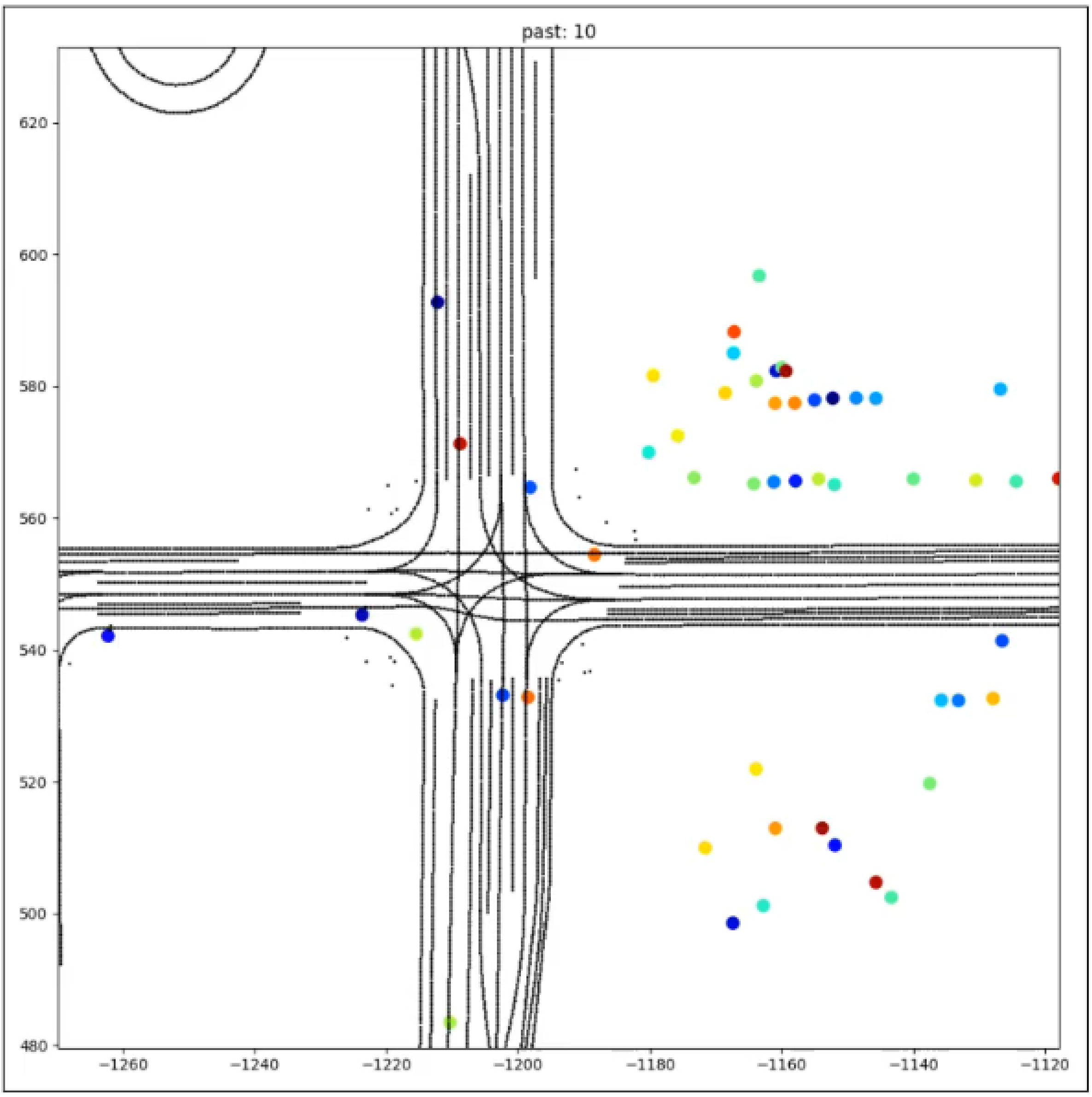


Figure 5.1: Scene at time step 0.

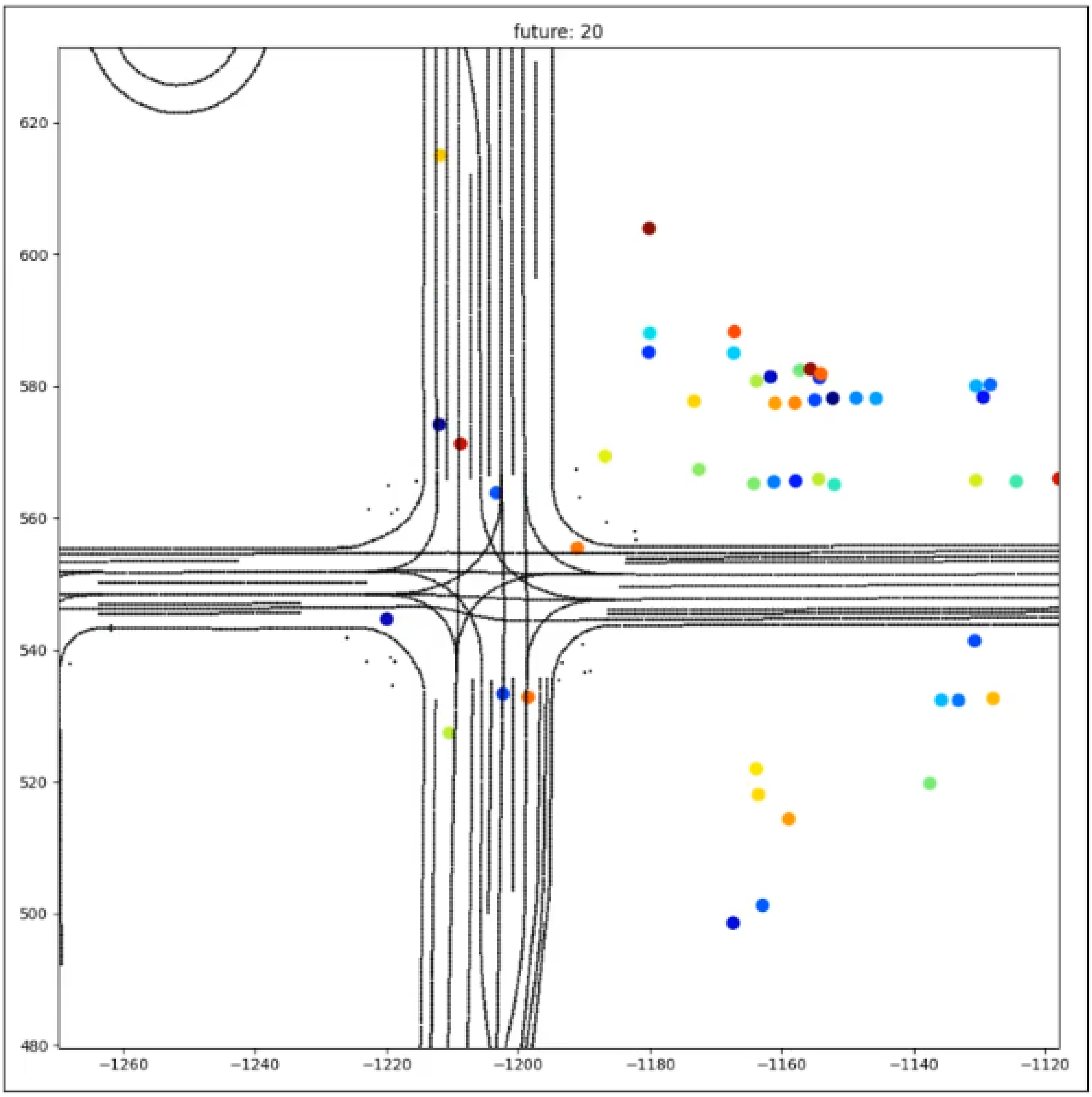


Figure 5.2: Scene at time step 20.

Figure 4.1, 4.2: Example of one scene at two different time steps.

The dataset is split into training, validation and test set, corresponding to 70%, 15%, and 15% of the data. The test set is not publicly available, so in this project, the validation set is split in half into a validation and a test set. Our pre-processing step includes a normalization step where features are centered about the origin so as to stabilize training. Feature extraction is one of the main routes of experimentation in our project design; our pre-processing step rasterizes each segment's context and determines what features are most relevant for each model. Specifically in our work, we included most object and roadgraph features, choosing to omit less pertinent features such as object size and sparse roadsign features such as traffic lights and their states.

5.2 Methods

5.2.1 Model Overview

Our approach was primarily inspired by the works mentioned in section 2. We combine both the modular approach described in the TNT and DenseTNT works with a simpler convolutional approach that was able to perform on par with more complex, state-of-the-art models. However, unlike the TNT and DenseTNT works, we chose to only output a single goal and trajectory for each object, omitting an additional trajectory ranking module. Thus, our final model consists of two modules: a goal prediction module and a trajectory prediction module. We extract relevant features as described above and vectorize the provided information for use as the input to each model in our pipeline. This approach allows us to identify and select features that are more relevant to a given module, reducing irrelevant information and the complexity of our overall model. The motion prediction task, as specified by the Waymo challenge, evaluates performance by requiring models to predict trajectories for up to eight objects in a given scenario. Therefore, both our goal and trajectory modules generate outputs for eight different objects at a time in a given scenario.

5.2.2 Long Short-Term Memory

The LSTM (Long Short-Term Memory) model is a type of recurrent neural network (RNN) architecture that has shown promising results in sequence modeling and prediction tasks, making it applicable to the Waymo Motion Prediction Challenge.

In the context of the challenge, the LSTM model can be utilized to capture the temporal dependencies and patterns present in the historical agent tracks provided for the past 1 second. By leveraging these temporal patterns, the LSTM model aims to predict the future positions of up to 8 agents for 8 seconds into the future.

The LSTM model consists of memory cells that can store information over long sequences and selectively update or forget that information based on the input data. This allows the model to effectively capture long-term dependencies and handle the challenges posed by sequential data.

The architecture of an LSTM model typically includes input, forget, and output gates, which regulate the flow of information through the memory cells. The input gate determines how much new information is added to the memory cells, the forget gate controls which information is discarded, and the output gate determines the information that is output from the memory cells.

To train the LSTM model for the Waymo Motion Prediction Challenge, the historical agent tracks and corresponding map data can be used as input sequences. The model can be trained using backpropagation through time, where the gradients are propagated through the time steps to update the model's parameters.

During the prediction phase, the LSTM model takes the historical agent tracks as input and generates predictions for the future positions of the agents. These predictions can be obtained by feeding the model's output back as input in an autoregressive manner, allowing the model to generate a sequence of predictions for multiple time steps into the future.

The LSTM model's ability to capture long-term dependencies and handle sequential data makes it suitable for predicting the future motion of road agents in the Waymo Motion Prediction Challenge. By effectively learning the temporal patterns from the historical agent tracks, the LSTM model can generate accurate trajectory predictions and contribute to the development of safer and more efficient self-driving systems.

Long Short-Term Memory (LSTM) blocks are usually used in sequential/temporal applications and their structure can be seen in figure 5.1. Their recurrent structure makes it possible to better learn sequences of data, with the memory component storing pertinent information that can be referenced later in a sequence. Since our model contains 10 discrete states containing past context and 1 discrete state containing current context, which are temporally related, LSTM blocks seem to be a strong approach that can capture these dependencies. As seen in figure 5.3, the block in the middle does not only take information about the current time step (X_t) but it also retrieves information from all previous steps (X_1, X_2, ...,X_{t-1}). Hence, the outputs at time t (Y_t)depend on all the previous inputs in addition to the current input. For more details on the concepts behind LSTM, we refer to [Gers & Schmidhuber (2001), Gábor Melis & Blunsom (2019)].

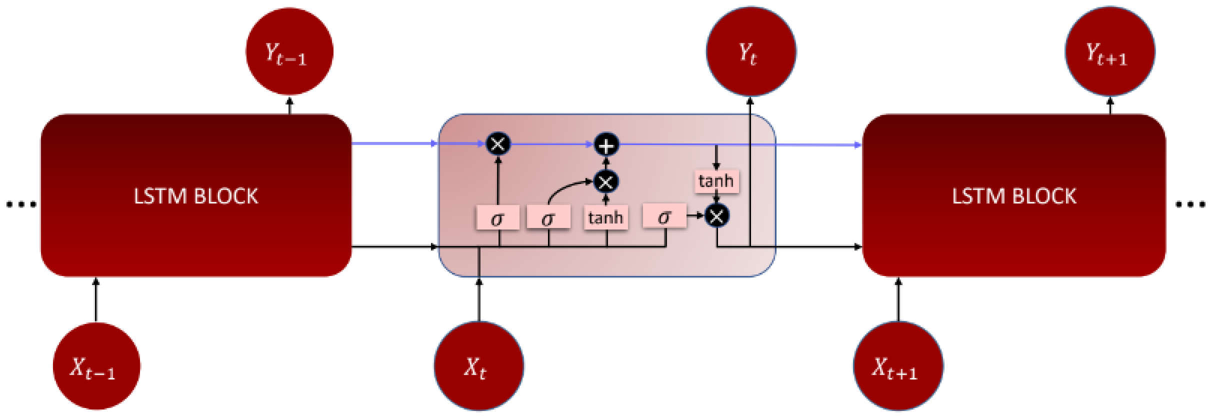


Figure 5.3: Internal LSTM cell structure.

5.2.3 Goal Prediction

The principle behind our goal prediction model is to estimate the endpoint of objects 8 seconds in the future provided with 1 second of past context. These estimates will then be used as anchors to predict object trajectories in the subsequent model. Though we introduced the LSTM chain above as having inputs and outputs in every block, we used an alternate structure in our goal prediction and trajectory completion modules. To estimate target positions 8 seconds into the future, we used the model provided in figure 5.4. The model can be deconstructed into a time-based component and a static component. The time-based LSTM chain sequentially processes inputs containing past states (including object velocity, yaw, and position). The chain contains eleven blocks (ten for the 1 second of past context, one for the current state), and only the output from the last block is used in the final dense layer.

In addition to the time dependent object information, each scene also contains static information such as the underlying roadgraph and other time-independent features. These features do not change over each timestamp and are instead processed independently. Static features are fed through four convolutional layers, each followed by a max pooling layer. To decrease the overfitting, we also add a dropout layer here. The output of the dropout layer is concatenated with the time-dependent outputs from the LSTM block. The resulting flattened vector is then fed into a dense layer and the output of the last dense layer is used as the endpoint predictions for all the vehicles we need to predict.

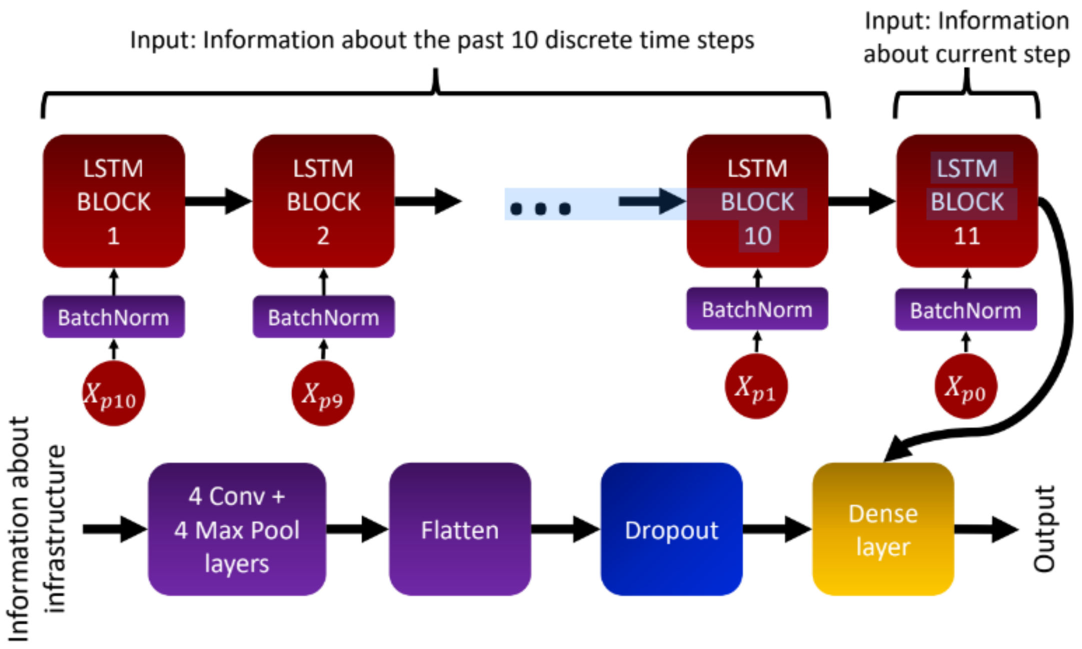

Input: Information about the past 10 discrete time steps
Input: Information about current step
LSTM BLOCK 1
LSTM BLOCK 2
LSTM BLOCK 10
LSTM BLOCK 11
BatchNorm
BatchNorm
BatchNorm
BatchNorm
X_{p10}
X_{p9}
X_{p1}
X_{p0}
Information about infrastructure
4 Conv + 4 Max Pool layers
Flatten
Dropout
Dense layer
Output

Figure 5.4: End point prediction model.

5.2.4 Trajectory Completion

The trajectory completion model receives as input a preprocessed scenario and couples it with the estimated endpoint given from the goal prediction model. The goal of this model is to predict 80 coordinates, ten for each second in the future, for each object conditioned on their goal. The first half of this model is similar to the goal prediction model given in figure 3 (the 'encoder'). Similar to the goal prediction model, time-dependent features are processed independently of static features prior to use in the decoder. The only difference between our previous model and the trajectory encoder is that encoder inputs now also contain the broadcasted predicted end points.

To predict the future points, we also define a 'decoder', given in figure 5.5. The decoder contains 8 LSTM cells, with each cell responsible for predicting one second of future motion. This architecture is similar to the cell structure described in section 4.2 with inputs being 0. The outputs of each LSTM should, when passed through a dense layer, output 10 points for each object, corresponding to one second of the eight-second prediction window. With all eight cells, we obtain the full 8-second predicted trajectories for all target objects.

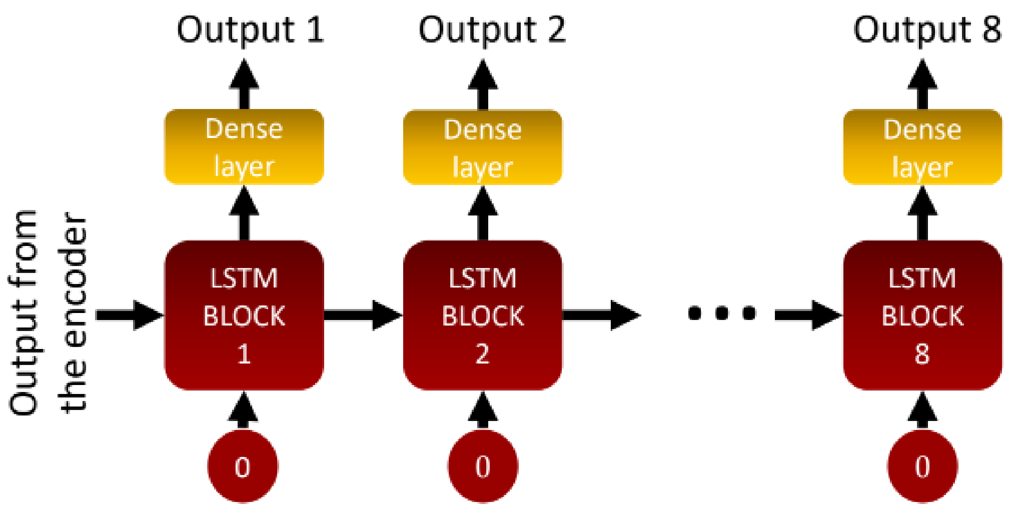


Figure 5.5: Trajectory prediction model.

Chapter 6

Results

This chapter contains the results obtained from above mentioned implementation.

6.1 Hyperparameters and Metrics

Hyperparameters for each module were found through a logarithmic range search. The best values found for each modules are specified in table 6.1. Mean square error was used as the loss function for both modules as it can be interpreted as the squared distance between the ground truth and the predicted coordinates, and the goal is to minimize this distance. For both modules, the loss function was minimized using an Adam optimizer with parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$. To evaluate model performance, we employ two metrics described in [Scott Ettinger (2021)], namely Final Displacement Error (FDE) and Average Displacement Error (ADE). The former evaluates the L2 norm between the prediction and the ground truth at the final time stemp, whereas the latter computes the average norm over all timestamps

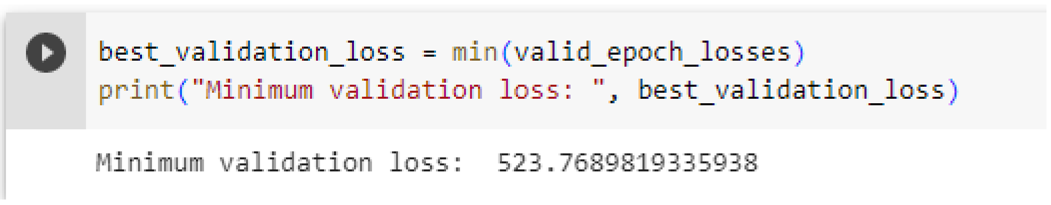
Table 6.1: Hyperparameters for the two modules.

Goal Prediction Module		
Learning Rate	**Batch Size**	**Dropout**
0.001	128	0.4

Trajectory Completion		
Learning Rate	**Batch Size**	**Dropout**
0.001	128	0.4

6.2 Goal Prediction

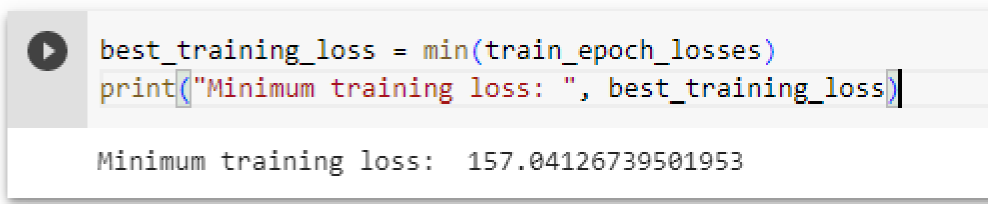
The loss for the training and the validation set for the goal prediction model can be seen in figure 6.13, where the best validation loss is 523.76 as shown in Fig 6.1 and the best training loss is 157.04 as shown in Fig 6.2, however the training loss was still declining rapidly when training was halted. This corresponds to an FDE of 22.8 as shown in Fig 6.4 and 12.5 meters as shown in Fig 6.6 in our validation and training sets, respectively. Furthermore, from figure 6.13 it is clear that the model overfits to the training data. The overfitting can also be seen by comparing the ground truth coordinates with the predictions made on the training and validation set as pictured in figure 6.13. Here, the predictions on the training set are more accurate than the predictions on the validation set.



```
best_validation_loss = min(valid_epoch_losses)
print("Minimum validation loss: ", best_validation_loss)

Minimum validation loss:  523.7689819335938
```

Figure 6.1: (a) Best Validation loss.

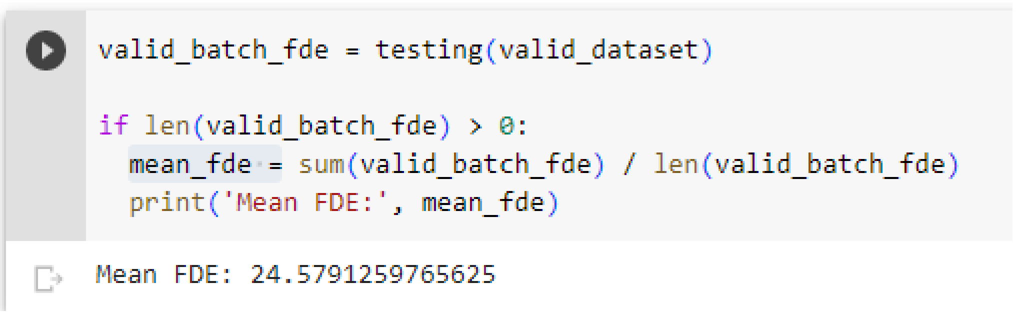


```
best_training_loss = min(train_epoch_losses)
print("Minimum training loss: ", best_training_loss)

Minimum training loss:  157.04126739501953
```

Figure 6.2: (b) Best Training loss.

Figure 6.1 & 6.2: Best Validation & Training loss for Goal Prediction Model.

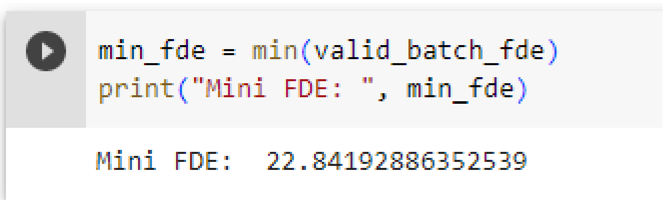


```
valid_batch_fde = testing(valid_dataset)

if len(valid_batch_fde) > 0:
  mean_fde = sum(valid_batch_fde) / len(valid_batch_fde)
  print('Mean FDE:', mean_fde)

Mean FDE: 24.5791259765625
```

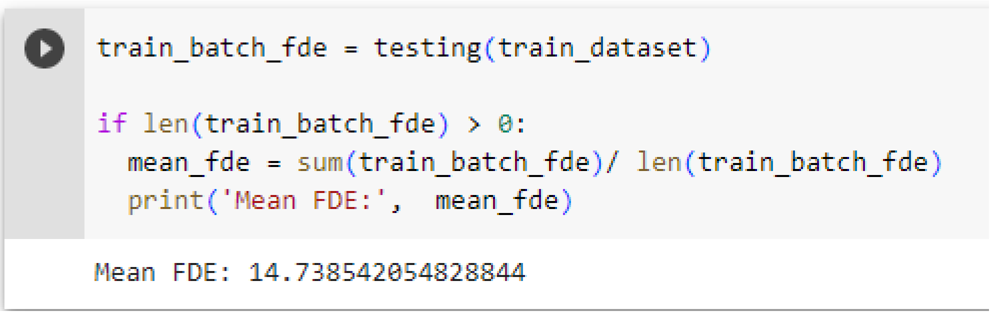
Figure 6.3: Mean FDE of Validation data for Goal Prediction Model.



```
min_fde = min(valid_batch_fde)
print("Mini FDE: ", min_fde)

Mini FDE:  22.84192886352539
```

Figure 6.4: Minimum FDE of Validation data for Goal Prediction Model.

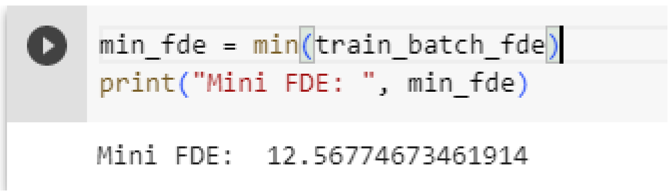


```
train_batch_fde = testing(train_dataset)

if len(train_batch_fde) > 0:
  mean_fde = sum(train_batch_fde)/ len(train_batch_fde)
  print('Mean FDE:',  mean_fde)
```

```
Mean FDE: 14.738542054828844
```

Figure 6.5: Mean FDE of Training data for Goal Prediction Model.



```
min_fde = min(train_batch_fde)
print("Mini FDE: ", min_fde)
```

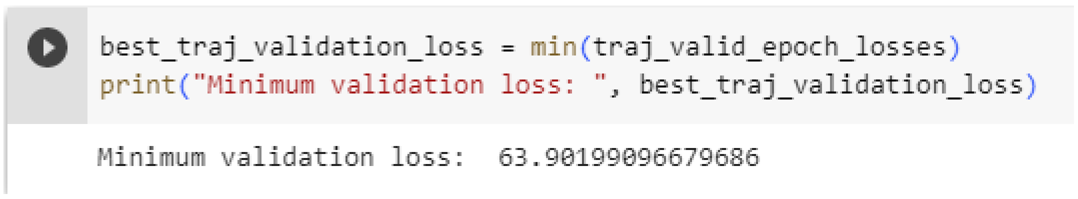
```
Mini FDE:  12.56774673461914
```

Figure 6.6: Minimum FDE of Training data for Goal Prediction Model.

One possible reason for our observed results is the limited regularization used in the model. To reduce variance, dropout could be applied not only to the static information but to the time dependent information as well. The dropout rate for the static information could also be increased in tandem with other regularization techniques. Given our computational and time restraints (with each training epoch taking 45 minutes for a single model), it was unfortunately not possible to make additional changes within the scope of this project. Another approach to reduce variance is to increase the size of the training dataset (e.g., with augmentation techniques).

6.3 Trajectory Completion

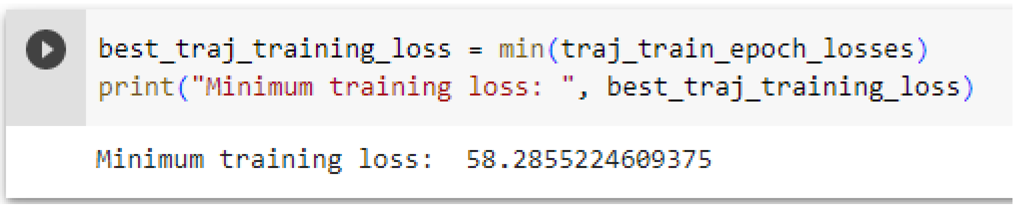
The loss for the trajectory completion model is shown in figure 6.14. Compared to the goal prediction model, both the training loss and the validation loss display a steeper descent, indicating better model generalization with small variance. This is expected as trajectory completion anchored on an endpoint is easier when compared against predicting goals as far as 8 seconds into the future. Observe that we used the ground truth values as end points in the training phase. The best validation loss for the trajectory completion is 63.90 as shown in Fig 6.7, and the best training loss is 58.29 as shown in Fig 6.8. This corresponds to an ADE of around 8.2 as shown in Fig 6.10 and 7.6 meters as shown in Fig 6.12 on the validation and training set respectively.



```
best_traj_validation_loss = min(traj_valid_epoch_losses)
print("Minimum validation loss: ", best_traj_validation_loss)
```

```
Minimum validation loss:  63.90199096679686
```

Figure 6.7: (a) Best Validation loss.

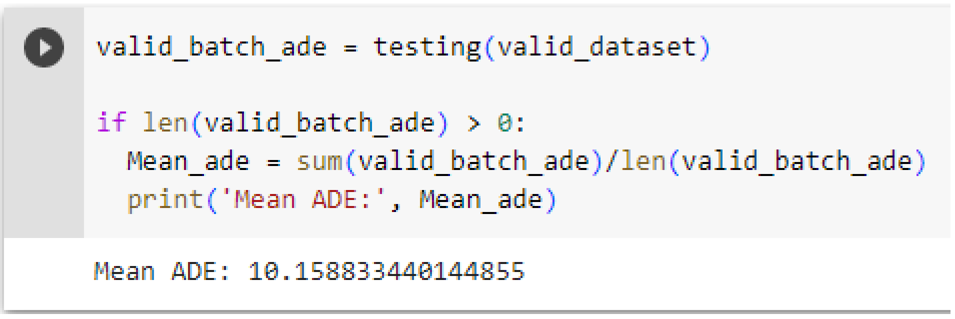


```
best_traj_training_loss = min(traj_train_epoch_losses)
print("Minimum training loss: ", best_traj_training_loss)
```

```
Minimum training loss:  58.2855224609375
```

Figure 6.8: (b) Best Training loss.

Figure 6.7 & 6.8: Best Validation & Training loss for Trajectory Completion Model.

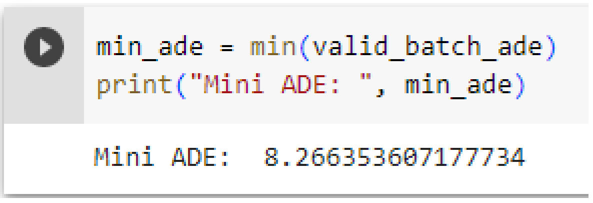


```
valid_batch_ade = testing(valid_dataset)

if len(valid_batch_ade) > 0:
  Mean_ade = sum(valid_batch_ade)/len(valid_batch_ade)
  print('Mean ADE:', Mean_ade)
```

```
Mean ADE: 10.158833440144855
```

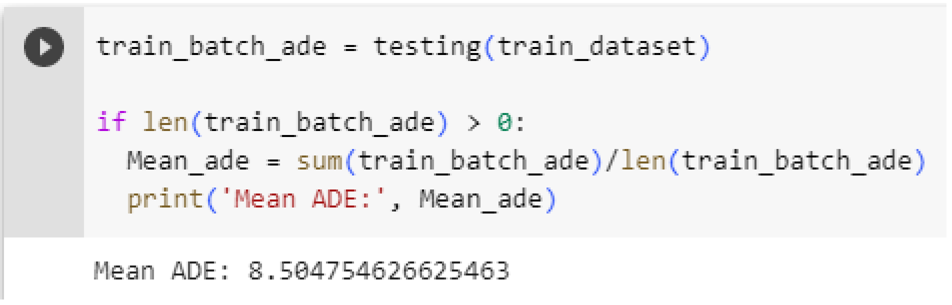
Figure 6.9: Mean ADE of Validation data for Trajectory Completion Model.



```
min_ade = min(valid_batch_ade)
print("Mini ADE: ", min_ade)
```

```
Mini ADE:  8.266353607177734
```

Figure 6.10: Minimum ADE of Validation data for Trajectory Completion Model.

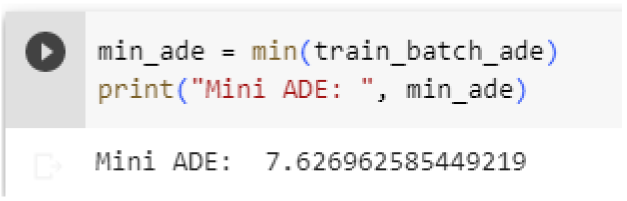


```
train_batch_ade = testing(train_dataset)

if len(train_batch_ade) > 0:
  Mean_ade = sum(train_batch_ade)/len(train_batch_ade)
  print('Mean ADE:', Mean_ade)
```

```
Mean ADE: 8.504754626625463
```

Figure 6.11: Mean ADE of Training data for Trajectory Completion Model.



```
min_ade = min(train_batch_ade)
print("Mini ADE: ", min_ade)
```

```
Mini ADE:  7.626962585449219
```

Figure 6.12: Minimum ADE of Training data for Trajectory Completion Model.

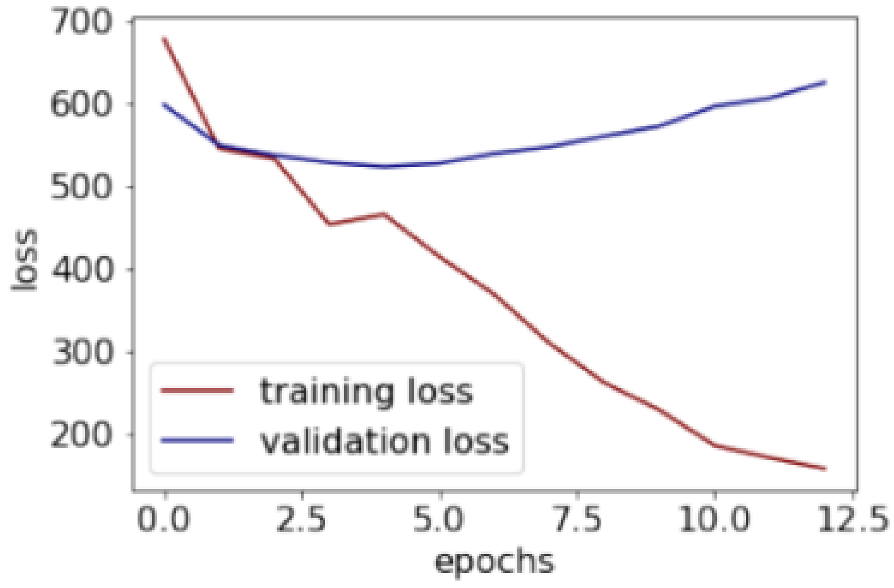


Figure 6.13: (a) Loss for the goal prediction model.

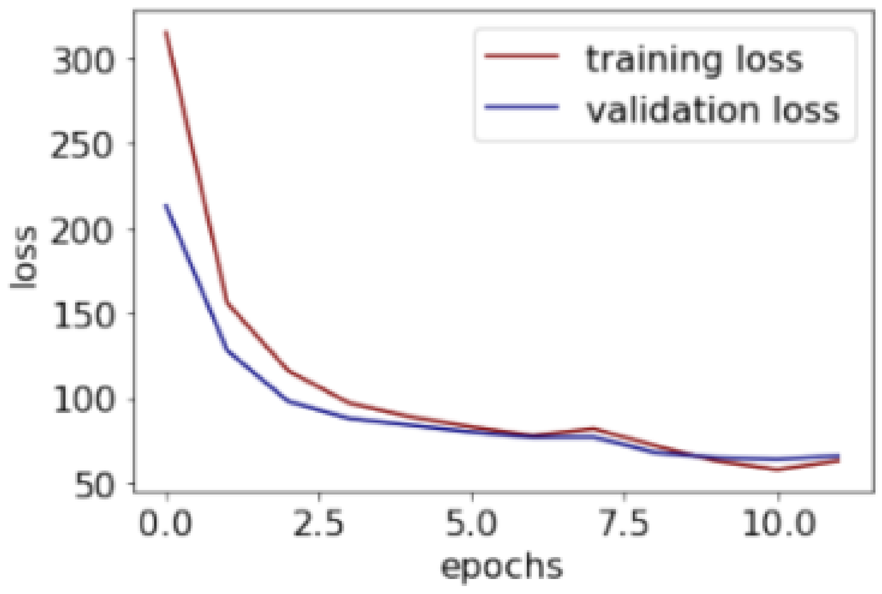


Figure 6.14: (b) Loss for the trajectory completion model.

Figure 6.13 & 6.14: The loss functions during training of the two models.

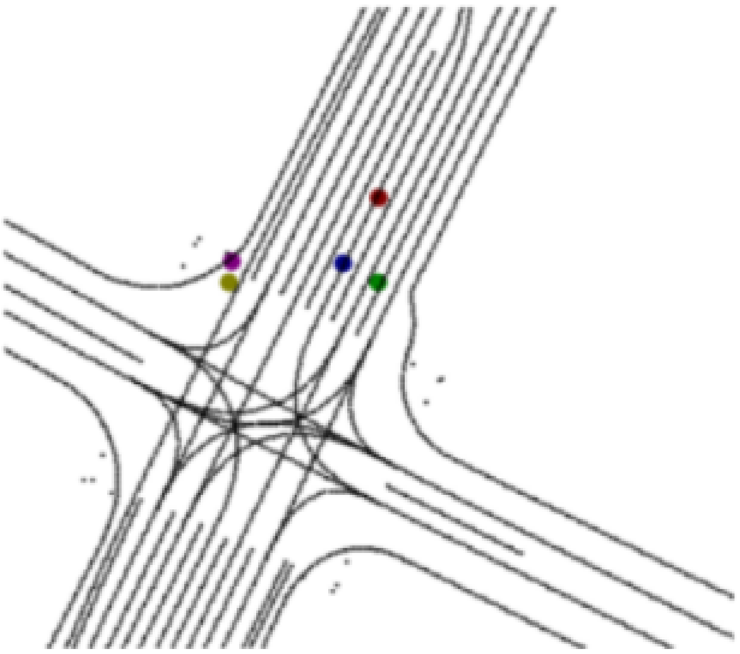


Figure 6.15: (a) GT for training example.

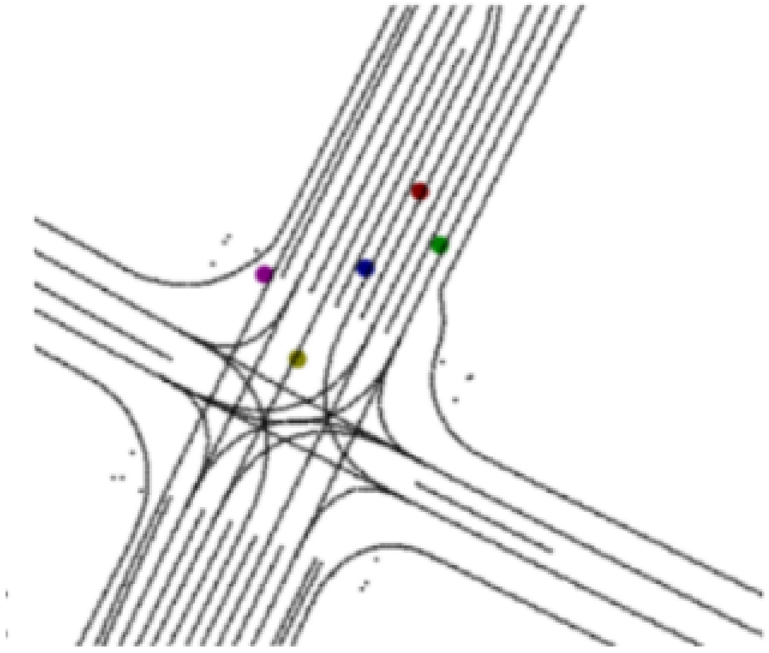


Figure 6.16: (b) Prediction for training example.

Figure 6.15 & 6.16: The ground truth (GT) positions and the predicted positions for a training example.

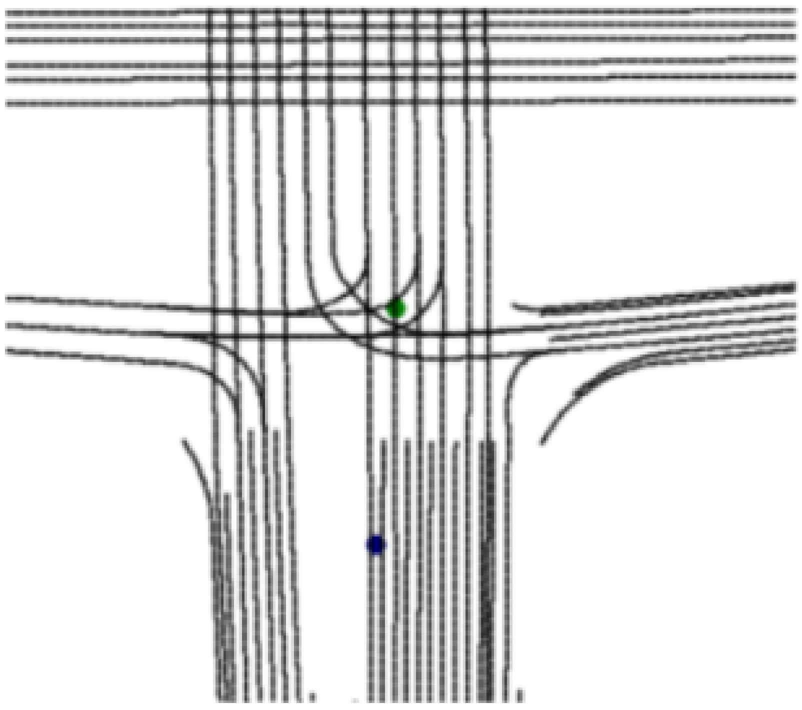


Figure 6.17: (c) GT for validation example.

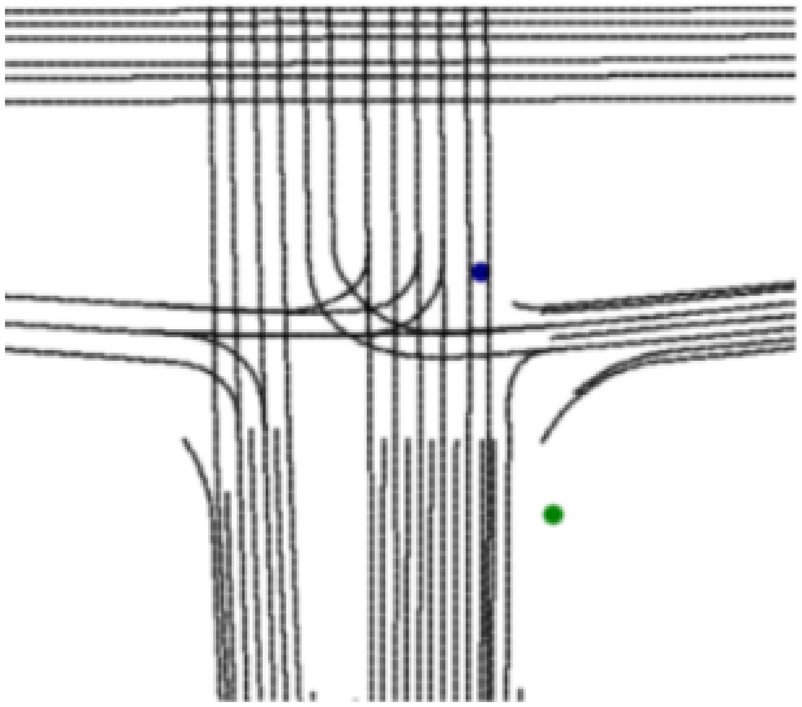
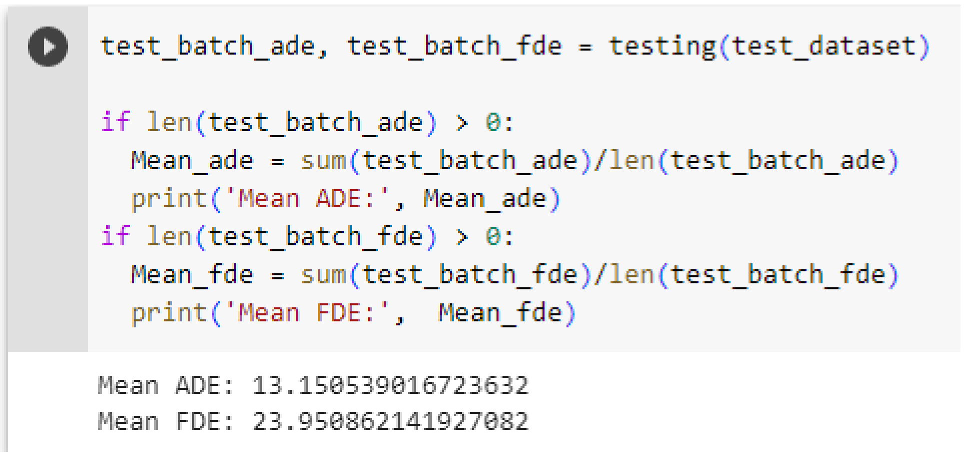


Figure 6.18: (d) Prediction for validation example.

Figure 6.17 & 6.18: The ground truth (GT) positions and the predicted positions for a validation example.

6.4 Combined model

The combined model was evaluated using a withdrawn test set. The endpoints were predicted using the goal prediction model and the rest of the trajectory was predicted using the trajectory completion model. The final displacement error (FDE) was 23.95, and the average displacement error (ADE) was 13.15 as shown in Fig 6.19.



```python
test_batch_ade, test_batch_fde = testing(test_dataset)

if len(test_batch_ade) > 0:
  Mean_ade = sum(test_batch_ade)/len(test_batch_ade)
  print('Mean ADE:', Mean_ade)
if len(test_batch_fde) > 0:
  Mean_fde = sum(test_batch_fde)/len(test_batch_fde)
  print('Mean FDE:',  Mean_fde)
```

```
Mean ADE: 13.150539016723632
Mean FDE: 23.950862141927082
```

Figure 6.19: Mean ADE & FDE of the above two combined Models on Test data.

Chapter 7

Conclusion and Future Work

We partitioned the task of motion prediction into two subtasks: goal prediction and trajectory completion. Our implementation of the goal prediction model had a very high variance and overfitted to the training data, which decreased the overall performance of the system. One way to improve upon on these results is to increase the regularization or the training data to reduce the overfitting of the goal prediction module. Our trajectory completion model had a far better performance and seems promising as a trajectory completion model. However, due to the low performance of the goal prediction model, our overall model's ADE and the FDE (13.15 and 23.95, respectively) are still higher than what would be acceptable in real world applications.

Potential avenues for future work include evaluating other model architectures (using 80 LSTM cells, one for each timestamp), further tuning the hyperparameters of the models with a greater focus on the goal prediction model, and evaluating the incorporation of other static and time-dependent context features.

Bibliography

A. Alahi, K. Goel, V. R. A. R. L. F.-F. & Savarese, S. (June 2016), '"social lstm: Human trajectory prediction in crowded spaces"'.

Anthony Hu, Zak Murez, N. M. S. D. J. H.-V. B. R. C. & Kendall., A. (2021), 'Fiery: Future instance prediction in bird'seye view from surround monocular cameras.', pp. 15273 – 15282.

A.Y., H. & D.F., W. (2015), 'Feature detection for vehicle localization in urban environments using a multilayer lidar', **17 (2)**, 420–429.

Balakrishnan Varadarajan, Ahmed Hefny, A. S. K. S. R. N. N. A. C. K. C. B. D. C. P. L. D. A. & Sapp., B. (2021), Multipath++: Efficient information fusion and trajectory aggregation for behavior prediction.

Coppola, R., . M. M. (2016), 'Connected car. acm computing surveys', **49**, 46.

DoT. (2016), 'Federal automated vehicles policy: Accelerating the next revolution in roadway safety. washington, dc: Us department of transport (dot).'.

Fernandes L.C., Souza J.R., P. G. S. P. S. D. M. C. P. M. K. R. M. A. & A., H. (2014), 'Carina intelligent robotic car: architectural design and applications', *Journal of Systems Architecture* **60 (4)**, 372–392.

Gers, F. & Schmidhuber, E. (2001), 'Lstm recurrent networks learn simple context-free and context-sensitive languages. ieee transactions on neural networks', **12(6)**, 1333–1340.

Gábor Melis, T. K. y. & Blunsom, P. (2019), 'Mogrifier lstm'.

H. Cui, V. Radosavljevic, F.-C. C. T.-H. L. T. N. T.-K. H. J. S. & Djuric, N. (2019), ‘“multimodal trajectory predictions for autonomous driving using deep convolutional networks”’, pp. 2090 – 2096.

Hang Zhao, Jiyang Gao, T. L. C. S.-B. S. B. V. Y. S. Y. S. Y. C. C. S. e. a. (2020.), ‘Tnt: Target-driven trajectory prediction.’.

Hendrickson, C., B. A. . M. Y. (2014), ‘Connected and autonomous vehicles 2040 vision.’.

Henggang Cui, Vladan Radosavljevic, F.-C. C. T.-H. L. T. N. T.-K. H. J. S. & Djuric., N. (2019), ‘Multimodal trajectory predictions for autonomous driving using deep convolutional networks’, pp. 2090 – 2096.

Holger Caesar, Varun Bankiti, A. H. L. S. V. V. E. L. Q. X. A. K. Y. P. G. B. & Beijbom., O. (2020), ‘nuscenes: A multimodal dataset for autonomous driving. in proceedings of the ieee/cvf conference on computer vision and pattern recognition’, *Autonomous vehicles* p. 11621–11631.

J., L. & S., T. (2010), ‘Robust vehicle localization in urban environments using probabilistic maps’, pp. 4372–4378.

J. Mercat, T. Gilles, N. E. Z. G. S. D. B. & Gil, G. P. (2020), ‘“multi-head attention for multi-modal joint vehicle motion forecasting”’, pp. 9638 – 9644.

Junru Gu, Q. S. & Zhao., H. (2021.), ‘Densetnt: Waymo open dataset motion prediction challenge 1st place solution’.

Kim, D., K. B. C. T. & Yi, K. (2017), ‘Lane-level localization using an avm camera for an automated driving vehicle in urban environments.’, **22**, 280–290.

Lyft. Lyft Level 5 (2022), `https://level-5.global/data/`. (Accessed on 19/01/2023).

M. Treiber, A. H. & Helbing, D. (2000), ‘“congested traffic states in empirical observations and microscopic simulations,”’, **E, vol. 62,**, 1795 – 1805.

Ming-Fang Chang, John Lambert, P. S. J. S. S. B. A. H. D. W. P. C. S. L. D. R. e. a. (2019), 'Argoverse: 3d tracking and forecasting with rich maps. in proceedings of the ieee/cvf conference on computer vision and pattern recognition', p. 8748–8757.

Moritz Werling, Julius Ziegler, S. K. & Thrun., S. (2010), Optimal trajectory generation for dynamic street scenarios in a frenet frame., *in* 'In 2010 IEEE International Conference on Robotics and Automation', IEEE, pp. 987 – 993.

N., O. (1979), 'A threshold selection method from gray-level histograms', **9 (1)**, 62–66.

Namhoon Lee, Wongun Choi, P. V. C. B. C. P. H. T. & Chandraker., M. (2017), 'Desire: Distant future prediction in dynamic scenes with interacting agents', pp. 336 – 345.

NHTSA (2020), 'Framework for automated driving system safety. washington, d.c., usa: National highway traffic safety administration (nhtsa), department of transportation (dot)'.

Rohde J., Jatzkowski I., M. H. & J.M., Z. (2016), 'Vehicle pose estimation in cluttered urban environments using multilayer adaptive monte carlo localization', pp. 1774–1779.

R.W., W. & R.M., E. (2017), 'Robust lidar localization using multiresolution gaussian mixture maps for autonomous driving', *International Journal of Robotics Research* **36 (3)**, 292–319.

SAE (2018), 'Surface vehicle recommended practice: (r) taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles'.

Scott Ettinger, Shuyang Cheng, B. C. C. L. H. Z. S. P. Y. C. B. S. C. Q. Y. Z. e. a. (2021), 'Large scale interactive motion forecasting for autonomous driving: The waymo open motion dataset.'.

Sergio Casas, W. L. & Urtasun, R. (2017), Learning to predict intention from raw sensor data, *in* 'In Conference on Robot Learning 2018 Automation Conference (ASP-DAC)', IEEE, pp. 629–634.

Shladover, S. E. (2018), 'Connected and automated vehicle systems.', *Journal of Intelligent Transportation Systems* **22**, 190–200.

Stepan Konev, K. B. & Sanakoyeu., A. (2021.), 'Motioncnn: A strong baseline for motion prediction in autonomous driving. in workshop on autonomous driving,'.

T. Salzmann, B. Ivanovic, P. C. & Pavone, M. (2020), '"trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data"', pp. 683 – 700.

Thrun S., B. W. & D., F. (2005), 'Probabilistic robotics'.

Tsung-Yi Lin, Priya Goyal, R. G. K. H. & Doll´ar., P. (2017), 'Focal loss for dense object detection.', pp. 2980 – 2988.

Veronese L., Aguiar, E. N. R. C. G. J. C. F. A. D. S. A. F. . O.-S. T. (2015), 'Re-emission and satellite aerial maps applied to vehicle localization on urban environments', pp. 4285–4290.

Waymo. Waymo Open Dataset. (2022), https://waymo.com/open/download/. (Accessed on 23/11/2022).

Wei Zhan, Liting Sun, D. W. H. S. A. C. M. N. J. K. H. K. C. S.-A. d. L. F. e. a. (2019), 'Interaction dataset: An international, adversarial and cooperative motion dataset in interactive driving scenarios with semantic maps'.

World Health Organization. Road traffic injuries. (2021), https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries. (Accessed on 16/08/2022).

Yang, S., W. W. L. C. & Deng, W. . (2018), 'Scene understanding in deep learning-based end-to-end controllers for autonomous vehicles.', **49**, 53–63.