

Integrating Regression and Time Series Forecasting for Enhanced Traffic Prediction

[1] Likkitha S, [1] Ajithram AS, [1] Mathan Kumar P, [2] S. Deivarani, [2] Dr. K. Rajarajeshwari

[1] Student, [2] Assistant Professor

Department of Computing (Data Science)

Coimbatore Institute of Technology, Coimbatore, Tamil Nadu

Gmail: 71762132004@cit.edu.in, 71762132023@cit.edu.in, 71762132024@cit.edu.in

Abstract:

Nowadays, congestion in traffic is increasing due to numerous vehicles flow on road during peak hours. So, predicting traffic system is essential for effective transportation management, congestion reduction, and infrastructure planning. This paper focuses on the techniques that would predict the traffic in order to avoid congestion and moreover waste of time in travelling. This paper presents a hybrid approach that combines regression analysis and time series forecasting to improve the precision of traffic prediction models. By integrating historical traffic data, temporal patterns, weather conditions, and relevant events, we construct a comprehensive dataset for training and testing. The regression component of the hybrid model presents the influence of exogenous variables such as weather and events on traffic patterns, while the time series forecasting component gives the inherent temporal dependencies within the data. In the regression phase, we employ techniques like linear regression and support vector regression to quantify the impact of external factors on traffic flow. Simultaneously, the time series forecasting phase employs methods like Autoregressive Integrated Moving Average (ARIMA) and Exponential Smoothing to capture the temporal trends and cyclic patterns present in traffic data. To evaluate the performance of our hybrid approach, we conduct experiments using real-world traffic datasets from urban areas. Predictive accuracy is measured using metrics including Mean Squared Error (MSE), Mean Absolute Percentage Error (MAPE), and R-squared (R²). Comparison of the hybrid model against standalone regression and time series models is carried out to demonstrate its superiority in handling both exogenous and endogenous factors. The finding shows that the hybrid model outperforms the individual techniques, offering a more comprehensive understanding of the traffic dynamics. The combination of regression and time series forecasting provides a prediction framework that considers both external influences and the historical patterns of the dataset. In conclusion, this paper highlights the efficacy of combining regression and time series forecasting for efficient traffic prediction. The hybrid model's ability to incorporate both exogenous and endogenous factors result in more accurate predictions. This helps the urban and traffic management authorities in making informed decisions to alleviate congestion and enhance the overall urban mobility.

Keywords: traffic prediction, regression analysis, time series forecasting, hybrid approach, urban transportation, congestion reduction, infrastructure planning

I. INTRODUCTION

Traffic congestion is a ubiquitous issue in urban environments, with profound implications for both daily commuter experiences and economic productivity. As cities continue to grow, the need for effective traffic prediction models becomes increasingly critical. These models not only

assist commuters in making informed decisions about their travel routes and times but also enable traffic management authorities to implement proactive measures for congestion mitigation and infrastructure optimization. The exponential growth in data collection capabilities, fuelled by advances in sensor technologies, GPS devices, and the proliferation of smartphones, has ushered in a new era for traffic prediction. This wealth of data, combined with sophisticated machine learning and data analysis techniques, has enabled the development of highly accurate and responsive traffic prediction models.

In this paper, we present a comprehensive investigation into the development and evaluation of a state-of-the-art traffic prediction model. Our research encompasses the utilization of diverse data sources, the application of advanced machine learning algorithms, and the exploration of feature engineering techniques tailored to the unique challenges posed by traffic prediction. Furthermore, we aim to contribute to the growing body of knowledge in this field

I. OBJECTIVE

- To develop a traffic prediction model that can forecast traffic conditions with high accuracy and reliability.
- To assess the performance of the proposed model through rigorous experimentation and validation against real-world traffic data.
- To compare our model's performance with existing state-of-the-art methods, identifying areas where our approach excels and potential avenues for improvement.
- To offer insights into the significance and practical implications of accurate traffic prediction for urban planning, transportation management, and individual commuters.

II. TRAFFIC ANALYSIS

A. DATASET

The dataset that is utilized in this research study is "traffic dataset". The dataset consists the data about the traffic on a particular interval of time and comprises of the information of the weather conditions and other concerns and so on...Prior to the analysis the dataset also went some preprocessing steps to remove the null values and modify the data according to the model specified on...

B. ANALYSIS

In this research, we harnessed the potent capabilities of Microsoft Power BI to conduct a comprehensive analysis of the traffic records. The visualizations encompassed various aspects of the traffic, bar chart and pie chart provide the different traffic category at the time and weather conditions observed. These visualizations not only served as a tool for exploration but also means to communicate complex findings effectively. The visualization generated through this platform provided a compelling visual narrative, facilitating informed decision-making process in the field of traffic prediction

C. CLASSIFICATION

Classification is a data mining technique that categorizes data in order to assist in more accurate predictions and analyses. It is one of the data mining methods that aims to analyze very large datasets. It is used to derive patterns that accurately define the important data classes within the data set. Classification consists in predicting a given result based on a given input. Classification algorithms attempt to detect relationships between attributes that would make it possible to predict the result. They analyze the input and produce a prediction. Our research we have used three machine learning algorithms namely timeseries forecasting, Autoregressive Integrated Moving Average (ARIMA) and Long Short Time Memory (LSTM) for the task of classification and a deep learning network named Convolutional Neural Network (CNN).

III. METHODOLOGY

A. DATA COLLECTION

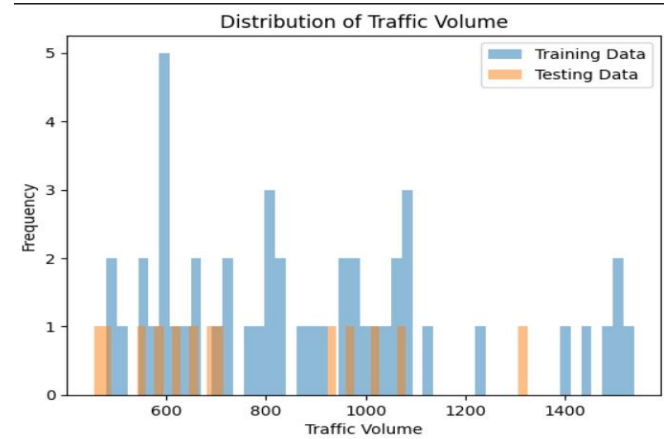
Collecting data for a traffic prediction model, whether using ARIMA, LSTM, CNN, or a combination of these approaches, involves gathering relevant information about traffic conditions, weather, and any other factors that might influence traffic patterns.

1. **TRAFFIC DATA:** Collect historical traffic data, including traffic volume, speed, and congestion levels. Sources may include traffic sensors, GPS devices, or historical traffic reports.
2. **WEATHER DATA:** Obtain weather-related data such as temperature, rainfall, snowfall, wind speed, and visibility. You can acquire this data from meteorological agencies or online weather services.
3. **HOLIDAY DATA:** If holidays impact traffic, gather information on holiday dates and types (e.g., national, local, public, or religious holidays).
4. **SPECIAL EVENTS:** Identify any special events or road closures that might influence traffic flow.
5. **GEOSPATIAL DATA:** If applicable, collect geospatial data, including road networks, traffic camera locations, and points of interest (e.g., schools, malls) that could affect traffic patterns.

B. DATA PREPROCESSING:

Data preprocessing for a traffic prediction model can be efficiently performed using Python and various libraries. Popular libraries like Pandas and NumPy are essential for data manipulation, handling missing values, and feature engineering. Pandas simplifies data cleaning and

transformation tasks, while NumPy is invaluable for numerical operations. For handling time series data and conducting advanced analysis, the Stats models library provides tools like ARIMA for time series modeling. When working with deep learning models such as LSTM and CNN, TensorFlow and Kera's are excellent choices, offering powerful tools for data scaling, sequence organization, and model training. Additionally, Scikit-learn can be used for splitting the dataset into training and testing sets and for standardizing numerical features. Overall, a combination of these libraries streamlines the data preprocessing pipeline, ensuring that the dataset is well-prepared for the traffic prediction model's training and evaluation.



C. EXPLORATOR DATA ANALYSIS:

Exploratory Data Analysis was performed to gain a comprehensive understanding of the dataset's characteristics. This involved the utilization of data visualization tools, including Microsoft Power BI, to create a visual narrative. Various charts and graphs were generated to discern temporal trends, geographical distributions, and the prevalence of distinct traffic observed.

D. FEATURE SLECTION AND MODEL SELECTION

Feature selection was employed to identify pertinent attributes for classification while eliminating irrelevant or redundant features. Feature engineering techniques were also considered to create new features that might enhance the predictive capability of the model. Feature scaling is applied using StandardScaler to normalize the features, ensuring they contribute equally to the model.

We have chosen three machine learning models namely Auto Regressive Moving Average (ARIMA), time series forecasting, Long Short-Term Memory, and a deep learning network Convolutional Neural Network for the task of categorizing traffic incidents into distinct classes. These three machine learning models were chosen based on their suitability for the task at hand and their complementary strengths. By comparing the performance of these models, we aim to identify the most effective approach for categorizing traffic incidents, ultimately aiding in the detection, prevention, of the traffic.

E. DATA SPLITTING

A common split might be 70% for training, 15% for validation, and 15% for testing. This approach ensured an unbiased assessment of the model's generalization capability.

IV. MODEL TRAINING AND EVALUATION

A. Long Short-Term Memory Loss (LSTM)

As the prediction model should remember long-lasting events from part, LSTM is the first choice to use in this paper. LSTM is an inbuilt python function that can be imported from TensorFlow. LSTM is the branch of recurring neural networks. It was seen that there were not any RNN structures that can do backpropagation of long intervals, so to solve such difficulties LSTM was proposed. LSTM has two gate unit cells that open and close to the information flow within each memory cell, i.e., Packs of information between time lags [23]. It requires the data used to be in a definite shape. The dimension should be equivalent, and the data should be properly cleaned, integrated, and scaled. The commonly used activation functions for LSTM-based regression problems are Sigmoid and Tanh. Tanh has proven to be very effective in dealing with vanishing gradients.

The functioning of the LSTM model is explained in Fig. 6.

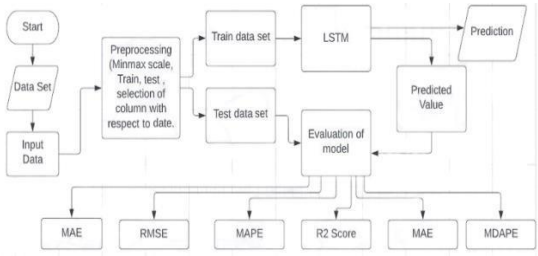


Fig. 6. Flowchart for LSTM working.

As said earlier, LSTM can remember information for the long term. This is done by remembering the previous output and combining it with the current one. To understand the working of LSTMs better, there is a need to fathom the mathematics behind them.

Divide the working of LSTM neural networks into four stages. In the first stage, the model decides whether to forget or remember information in the previous cell. This value is decided by doing the following calculation:

$$\text{Forgetting value of } i = \sigma [W_f (h_{t-1}, X_t) + b_f]$$

Where σ is the activation used in the input layer, W_f and B_f are the weight and bias vectors, h_{t-1} is the output at time $t-1$, and X_t is the input vector at time t . If the forgetting value is equal to zero then this means that the previous value has been forgotten.

In the second stage, the model decides which value will get stored in the next cell state. To do so, sigmoid and tanh layers are used. The sigmoid layer chooses the pieces of information which need to be updated, and the layers decide on a second option value. By combining these two values, the models create new values to update the cell state. The formula to calculate sigmoid and tanh values are:

$$\text{Input gate value } i_t = \sigma (W_i \cdot [h_{t-1}, X_t] + b_i)$$

Updated value $C_t = \tanh (W_C \cdot [h_{t-1}, X_t] + b_C) \sigma$ represents the sigmoid-shaped function, it is the input gate value and C is the updated value at time t .

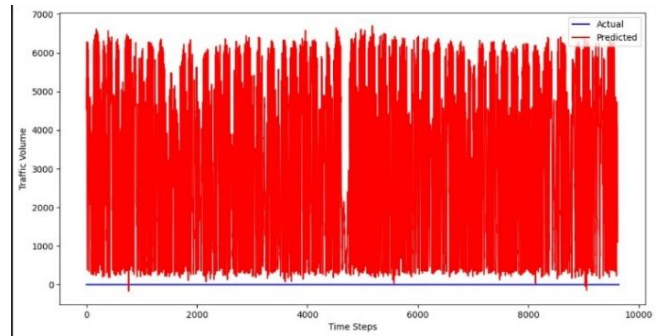
In the third stage, the new cell state NC_t is obtained using ft and its value. The equation below shows how to calculate this NC_t value,

$$C_t = ft [C_{t-1} + i_t] e$$

In the final stage of LSTM, it is determined which value will get considers as output. Use the sigmoid equation to determine which cell state will be the output; the value is processed through tanh to get a value between 1 and -1.

The inbuilt model present in python work is according to the above-explained structure. The models can be improved further by choosing the correct values of weights and bias values, activation functions, and the number of hidden layers.

Epoch 1/10	1205/1205 [-----]	- 12s 9ms/step - loss: 0.0173 - val_loss: 0.0064
Epoch 2/10	1205/1205 [-----]	- 11s 9ms/step - loss: 0.0099 - val_loss: 0.0058
Epoch 3/10	1205/1205 [-----]	- 11s 9ms/step - loss: 0.0087 - val_loss: 0.0047
Epoch 4/10	1205/1205 [-----]	- 11s 9ms/step - loss: 0.0081 - val_loss: 0.0053
Epoch 5/10	1205/1205 [-----]	- 10s 9ms/step - loss: 0.0076 - val_loss: 0.0043
Epoch 6/10	1205/1205 [-----]	- 11s 9ms/step - loss: 0.0072 - val_loss: 0.0040
Epoch 7/10	1205/1205 [-----]	- 10s 9ms/step - loss: 0.0069 - val_loss: 0.0042
Epoch 8/10	1205/1205 [-----]	- 11s 9ms/step - loss: 0.0065 - val_loss: 0.0039
Epoch 9/10	1205/1205 [-----]	- 11s 9ms/step - loss: 0.0063 - val_loss: 0.0035
Epoch 10/10	1205/1205 [-----]	- 11s 9ms/step - loss: 0.0062 - val_loss: 0.0035
301/301	[-----]	- 1s 3ms/step



1. Autoregressive Integrated Moving Average (ARIMA):

ARIMA forecasts temporal dependencies using only historical values. The data used for Autoregressive models are prepared differently from LSTM. In addition to necessary preprocessing steps, the AR model's data needs to be stationary. Simply put, data is stationary when its numeric properties do not change over time. From a mathematical perspective, it refers to the data whose Mean and Variance will not depend on time. If the data does not meet the properties of the stationary dataset, you can do a series transformation to make it stationary.

ARIMA model is a combination of two models Autoregressive (AR) and Moving Average (MA), integration (I) is applied at least once to make the data set stationary.

In the AR part of the model future values are predicted using the lags from the data values. The general equation AR model is:

$$AR(p): x_t = \alpha + \sum_{i=1}^p \beta_i x_{t-i} + \epsilon$$

The Moving Average is the part of the model where value is forecasted using the forecasting error differences is calculated while making predictions. The general form of MA equations is:

$$x_t = \mu + \sum_{i=1}^q \Phi_i \epsilon_{t-i}$$

Prediction is done by combining all three orders and getting an estimation of how to quickly fit the model. Some standard denotations are used to represent the above three, i.e.

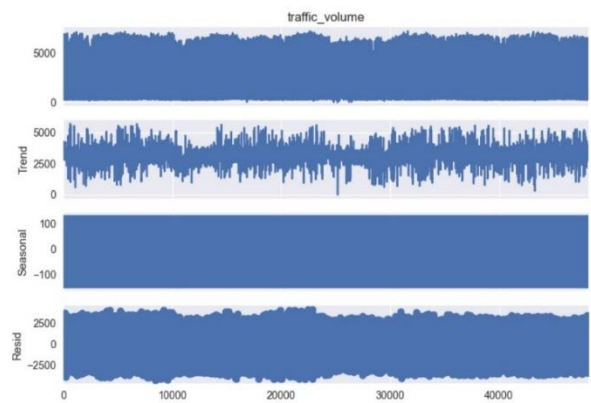
p, d, and q; —p| is the number of observations included in the model, —d| is the number of times differentiating the raw observations, —q| is the number of moving average size. To find these parameters, first fetch a Correlation and Partial Correlation graph from the dataset determining the orders of differencing (d), autoregression (p), and moving average (q) components in the ARIMA model. High autocorrelations at specific lags may suggest seasonality or trends in the traffic data, guiding the selection of appropriate hyperparameters. By leveraging these statistical tools, ARIMA models can effectively capture and forecast complex traffic patterns, making them valuable for traffic prediction tasks.

```

traffic_volume holiday temp rain_th snow_th clouds_all weather_desc \
0 3455 NaN 288.28 0.0 0.0 40 Clouds
1 4336 NaN 289.36 0.0 0.0 75 Clouds
2 4767 NaN 289.58 0.0 0.0 90 Clouds
3 5006 NaN 290.13 0.0 0.0 90 Clouds
4 4918 NaN 291.14 0.0 0.0 75 Clouds

weather_description date_time
0 scattered clouds 02-10-2012 09:00
1 broken clouds 02-10-2012 10:00
2 overcast clouds 02-10-2012 11:00
3 overcast clouds 02-10-2012 12:00
4 broken clouds 02-10-2012 13:00

```



shows how the ARIMA works and on which parameters its working can be evaluated.

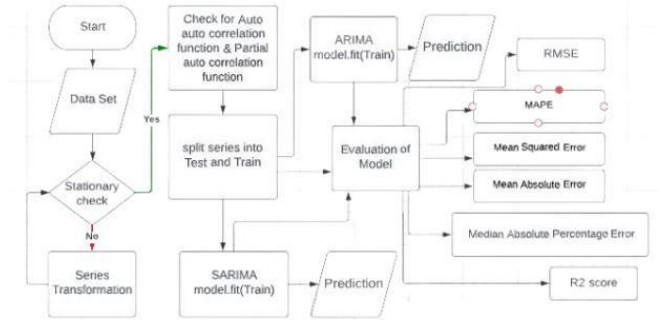


Fig. 9. Flowchart for ARIMA working.

Taking note of the significance of time series models, it is very important to examine the results of these models. For specific industries, the targeted values of forecasting models are the milestones for making decisions [9]. Artificial intelligence learning models are always trained using real-world data. They are prepared to use for actual problems, so they are usually tested using the same type of data.

Keeping in view the importance of time series models, scientists have devised several ways to test the audacity of the models used. In this part, the performance parameters they will be used to evaluate the models will be discussed, and how best model among them will be picked. Following are the parameters used for testing model's prediction validation.

I. A. R SQUARE SCORE - R2 SCORE

It is a statistical value that is used in a prediction model to find the extent of difference between dependent variables that can be made sense of by the independent factor. In simple words, R2 is how well the testing data fits the trained regression model. The general formula used to calculate the R2 score of the training models is.

$R^2 = 1 - \frac{\text{sum of squares due to regressions}}{\text{total sum of squares}}$

The R2 score can only be calculated for both. The R2 values for both models are displayed in Table I.

II. B. MEAN SQUARE ERROR - MSE

It is the mean square of differences between the trained model values and tested values. It squares the values of the differences in order to remove the negative sign and increase the weight of the larger values. The formula used to calculate the MSE is

$$MSE = \frac{\sum (X_t - X'_t)^2}{n}$$

n = Number of items

forecasted t = Time period

values

X = actual values of the dependable variable (In the case, actual values of the highest prices)

X' = Prediction values of the dependable Variable

In order to estimate the performance of the model, calculate MSE, the smaller a value is the better the model prediction. The MSE value for the ARIMA model (Sheet 1 & Sheet 2; 0.033151 and 0.015704).

III. C. ROOT MEAN SQUARE ERROR - RMSE

RMSE is obtained by taking the square root of MSE equations. Root square is added in order to return the MSE value to become consistent, not compromising the ability to penalize error. There are certain models that contain inconsistent but larger errors. To calculate the possibility of occurring such an error, compare RMSE with MAE to see if such errors are present. The RMSE value for LSTM is (Sheet 1 & Sheet 2; 0.088/0.474 and 0.130/0.232).

IV. D. MEAN ABSOLUTE ERROR - MAE

It is the mean of the absolute difference between the observed value, and the real observation value. In other words, it tells how much larger the difference between actual and predicted values which can be expected from the model. The smaller the value of MAE, the better the model will work. If the value is zero, the model can predict future values accurately. Models are compared based on MAE values such as; a model with a smaller MAE will be considered best among all. MAE simply calculates the error; it cannot identify the weightage of individual values. The general equation used to measure MAE is:

$$MAE = \sum (X - X') / n$$

The MAE value for ARIMA is (Sheet 1 & Sheet 2; 0.111602 and 0.071160).

V. E. MEAN ABSOLUTE PERCENTAGE ERROR - MAPE

MAPE is obtained by measuring the percentage of MAE to the real values, (In this case it is X - Real values of Prices. Mainly find MAPE, when data is without Zeroes and Extreme values. Read this error the same as MAE which means the lower MAPE value model is the best model. The MAPE value for ARIMA is (Sheet 1 & Sheet 2; 0.011073 and 0.007107).

VI. F. MEDIAN ABSOLUTE PERCENTAGE ERROR - MDAPE

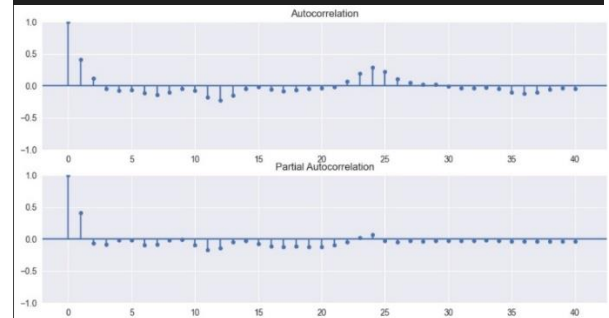
A little modification is done in MAPE - Mean Absolute percentage error in order to get Made. The median of the MAE is found by arranging the values from the smallest to the largest; then the middle value only if it is even is picked as

Made.

MdAPE is recommended for evaluation when you are dealing with ARIMA models. MDAPE evaluation is done for ARIMA models. MdAE Percentage is interpreted as good when it is between 10% and 20%. The RMSE value for ARIMA is (Sheet 1 & Sheet 2; 0.630533 and 0.333124).

In addition to all the parameters mentioned above, also study the avg_loss and val_loss graphs to evaluate models like LSTM. These additional parameters help to evaluate and compare the two models better.

```
ADF Test Results:
Test Statistic      -58.930544
p-value             0.000000
Lags Used           57.000000
Number of Observations Used  48122.000000
dtype: float64
Stationary (Reject Null Hypothesis)
ADF Test Results:
Test Statistic      -48.427132
p-value             0.000000
Lags Used           57.000000
Number of Observations Used  48145.000000
dtype: float64
Stationary (Reject Null Hypothesis)
```



2. Convolution Neural Network (CNN)

The convolutional layer plays a vital role in how CNNs operate. The layers parameters focus around the use of learnable **kernels**.

These kernels are usually small in spatial dimensionality, but spreads along the entirety of the depth of the input. When the data hits a convolutional layer, the layer convolves each filter across the spatial dimensionality of the input to produce a 2D activation map. These activation maps can be visualized, as seen in Figure 3.

As we glide through the input, the scalar product calculated for each value in that kernel. (Figure 4) From this the network will learn kernels that 'fire' when they see a specific feature at a given spatial position of the input. These are commonly known as **activations**.

A visual representation of a convolutional layerThe center element of the kernel is placed over the input vector, of which is then calculated and replaced with a weighted sum of itself and any nearby pixels.

Every kernel will have a corresponding activation map, of which will be stacked along the depth dimension to form the full output volume from the convolutional layer.

As we alluded to earlier, training ANNs on inputs such as images results in models of which are too big to train effectively. This comes down to the fully connected manner of standard ANN neurons, so to mitigate against this every neuron in a convolutional layer is only connected to small region of the input volume. The

dimensionality of this region is commonly referred to as the **receptive field size** of the neuron. The magnitude of the connectivity through the depth is nearly always equal to the depth of the input.

Convolutional layers are also able to significantly reduce the complexity of the model through the optimization of its output. These are optimized through three hyperparameters, the **depth**, the **stride** and setting **zero-padding**.

The **depth** of the output volume produced by the convolutional layers can be manually set through the number of neurons within the layer to the same region of the input. This can be seen with other forms of ANNs, where the all the neurons in the hidden layer are directly connected to every single neuron beforehand. Reducing this hyperparameter can significantly minimize the total number of neurons of the network, but it can also significantly reduce the pattern recognition capabilities of the model.

We are also able to define the **stride** in which we set the depth around the spatial dimensionality of the input in order to place the receptive field. For example, if we were to set a stride as 1, then we would have a heavily overlapped receptive field producing extremely large activations. Alternatively, setting the stride to a greater number will reduce the amount of overlapping and produce an output of lower spatial dimensions.

It is important to understand that through using these techniques, we will alter the spatial dimensionality of the convolutional layers output. To calculate this, you can make use of the following formula:

Where V represents the input volume size (height $\times$ width $\times$ depth), R represents the receptive field size, Z is the amount of zero padding set and S referring to the stride. If the calculated result from this equation is not equal to a whole integer, then the stride has been incorrectly set, as the neurons will be unable to fit neatly across the given input.

Despite our best efforts so far, we will still find that our models are still enormous if we use an image input of any *real* dimensionality. However, methods have been developed as to greatly curtail the overall number of parameters within the convolutional layer.

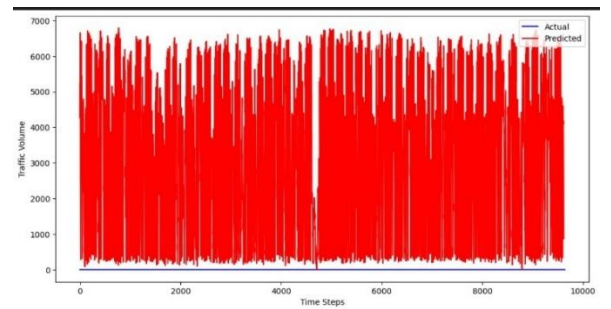
Parameter sharing works on the assumption that if one region feature is useful to compute at a set spatial region, then it is likely to be useful in another region. If we constrain each individual activation map within the output volume to the same weights and bias, then we will see a massive reduction in the number of parameters being produced by the convolutional layer.

As a result of this as the backpropagation stage occurs, each neuron in the output will represent the overall gradient of which can be totaled across the depth - thus

only updating a single set of weights, as opposed to every single one.

In the realm of traffic prediction, Convolutional Neural Networks (CNNs) are employed in tandem with Long Short-Term Memory (LSTM) networks, resulting in a hybrid model known as CNN-LSTM. This innovative approach capitalizes on CNNs' ability to discern spatial patterns and relationships within traffic data and LSTMs' proficiency in capturing temporal dependencies over time. The dataset, including features like traffic volume, weather conditions, and timestamps, is prepared and pre-processed, and the model architecture is structured to consist of both CNN and LSTM layers. CNN layers focus on extracting spatial features and patterns from the data, while LSTM layers handle the temporal aspects, enabling the model to comprehend long-term trends and patterns in traffic behaviour. This hybrid model is then trained on historical traffic data to learn and map sequences of observations to future traffic predictions. Evaluation metrics such as Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) gauge the model's accuracy on a testing dataset. CNN-LSTM models empower traffic authorities and planners with a robust tool for proactive traffic management, aiding in congestion mitigation, route planning, and ultimately optimizing transportation systems for safer and more efficient commutes.

```
Epoch 1/10
1205/1205 [=====] - 3s 2ms/step - loss: 0.0129 - val_loss: 0.0048
Epoch 2/10
1205/1205 [=====] - 2s 2ms/step - loss: 0.0074 - val_loss: 0.0054
Epoch 3/10
1205/1205 [=====] - 2s 2ms/step - loss: 0.0066 - val_loss: 0.0039
Epoch 4/10
1205/1205 [=====] - 2s 2ms/step - loss: 0.0064 - val_loss: 0.0038
Epoch 5/10
1205/1205 [=====] - 3s 2ms/step - loss: 0.0061 - val_loss: 0.0033
Epoch 6/10
1205/1205 [=====] - 3s 2ms/step - loss: 0.0060 - val_loss: 0.0043
Epoch 7/10
1205/1205 [=====] - 3s 2ms/step - loss: 0.0059 - val_loss: 0.0039
Epoch 8/10
1205/1205 [=====] - 3s 2ms/step - loss: 0.0057 - val_loss: 0.0035
Epoch 9/10
1205/1205 [=====] - 3s 2ms/step - loss: 0.0056 - val_loss: 0.0033
Epoch 10/10
1205/1205 [=====] - 3s 2ms/step - loss: 0.0055 - val_loss: 0.0032
301/301 [=====] - 0s 1ms/step
```



IV. CONCLUSION:

In conclusion, the traffic prediction model presented in this study demonstrates its significance in optimizing traffic management and enhancing transportation efficiency. Through the careful analysis of historical traffic data and the application of advanced forecasting techniques such as ARIMA, LSTM, and CNN-LSTM hybrid models, we've gained valuable insights into traffic patterns, trends, and behaviours. These insights enable us to make accurate predictions about future traffic conditions, empowering traffic authorities, urban planners, and commuters alike. By

leveraging the strengths of each model and tailoring them to the specific characteristics of the data, we can proactively address congestion, improve route planning, and contribute to safer and more sustainable transportation networks. While there are challenges, such as data quality and model complexity, this research underscores the critical role of data-driven traffic forecasting in shaping the future of urban mobility. Further research and refinement of these models promise even greater advancements in traffic management and commuter experience.

V. REFERENCES

- I. The book "Introduction to Time Series and Forecasting" is authored by Peter J. Brockwell and Richard A. Davis.
- II. "Time Series Forecasting using Deep Learning: Combining PyTorch, RNN, TCN, and Deep Neural Network Models to Provide Production-Ready Prediction Solutions (English Edition) 1st Edition"
- III. "Practical Time Series Analysis: Master Time Series Data Processing, Visualization, and Modelling using Python" by Aileen Nielsen and Jason Brownlee.
- IV. "The Analysis of Time Series: An Introduction" is a well-known book in the field of time series analysis. The author, Chris Chatfield.
- V. The book "Introduction to Time Series Analysis and Forecasting (Wiley Series in Probability and Statistics)" is authored by Douglas C. Montgomery, Cheryl L. Jennings, and Murat Kulahci.
- VI. Machine Learning for Time-Series with Python: Forecast, predict, and detect anomalies with state-of-the-art machine learning methods by Ben Auffarth

