“A Deep Learning Approach for Bottleneck Detection in IoT”

A Project work Submitted to KCES’s

Moolji Jaitha College, Jalgaon

(An Autonomous College Affiliated to KBCNMU Jalgaon)

For the compliance of work performed under

‘Research Promotion Scheme for Budding Researchers’

In the faculty of Science

By

Mohit Shambhunath Maurya - TYBSc(Computer Science)

Under the Guidance of

Mrs. Hemalata H. Patil

Department of Computer Science

Moolji Jaitha College (Autonomous), Jalgaon

March 2024

KCES's Moolji Jaitha College, Jalgaon

(An Autonomous College Affiliated to KBCNMU Jalgaon)

M. J. College Sponsored

Research Promotion Scheme for Budding Researchers

2023-2024

Certificate

This is to certify that, Mohit Shambhunath Maurya has completed his project work entitled '**A Deep Learning Approach for Bottleneck Detection in IoT**' for the Research Promotion Scheme for Budding Researchers in Department of Computer Science at Moolji Jaitha College (Autonomous), Jalgaon for the year 2023-24.

(Mr. P. M. Nikume) **(Mrs. Hemalata H patil)**

Committee Chairman **HOD Dept. of Computer Science**

DECLARATION

I hereby declare that the project entitled "**A Deep Learning Approach for bottleneck detection**" completed and written by me under the supervision of Mrs. Hemlata H patil . The present work is original and performed by me and previously not formed the basis for the award of any degree or diploma or other similar titles of this or any other University or examining body.

Mohit Shambhunath Maurya

Mrs. Hemlata H. Patil
Head of Dept. of Computer science

Place:

Date:

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my supervisor Mrs. Hemlata H Patil , for their invaluable guidance and expertise that played a pivotal role in shaping the trajectory of this research project. I extend my appreciation to my peers for their unwavering support and insightful feedback, and to my family and friends for their understanding and encouragement during the challenging phases of this endeavour. Special thanks to KCES Moolji Jaitha College for their crucial financial support, facilitating the execution of experiments and the acquisition of necessary resources. I also acknowledge the faculty members of at KCES Moolji Jaitha College for fostering an enriching academic environment. This project's completion is a testament to the collaborative effort of everyone involved, and I want to express my thanks to each person for their contributions, whether big or small. The encouragement received from various quarters has been a driving force in navigating obstacles and maintaining momentum. I deeply appreciate the collective effort that has made this project a reality.

Introduction

The increasing complexity of modern computer systems and networks has led to a growing concern regarding performance bottlenecks. A bottleneck refers to a point in a system where the flow of data or processes is significantly restricted, leading to suboptimal performance. Identifying and mitigating bottlenecks is crucial for enhancing the efficiency and overall functionality of systems. In the context of computer networks, software applications, or hardware architectures, bottlenecks can impede the timely processing of tasks and degrade overall system performance.

The purpose of this project is to address the challenge of bottleneck detection using a deep learning approach. Deep learning, a subset of machine learning, has demonstrated remarkable capabilities in pattern recognition and feature extraction, making it an ideal candidate for identifying complex patterns associated with bottleneck conditions. By leveraging the power of deep neural networks, we aim to develop a robust and accurate system for detecting bottlenecks in diverse computing environments.

Objectives of the Project:

1. Develop a deep learning model capable of identifying patterns indicative of bottleneck conditions in various computing systems.
2. Train the model using representative datasets from different types of systems to enhance its generalisation capabilities.
3. Evaluate the performance of the deep learning model in real-world scenarios and compare it with existing bottleneck detection methods.
4. Investigate the potential impact of bottleneck detection on system performance and explore strategies for bottleneck mitigation.

Significance of the Project:

The efficient functioning of computer systems is vital for numerous applications, including cloud computing, data centres, and edge computing. The detection and timely resolution of bottlenecks contribute to the optimisation of resource utilisation, reducing latency, and ensuring smooth operation of applications. This project's

outcomes have the potential to enhance the reliability and performance of a wide range of computing systems, positively impacting industries reliant on high-performance computing.

Scope of the Project:

This project focuses on the development and evaluation of a deep learning-based solution for bottleneck detection. The scope encompasses various computing environments, including but not limited to cloud-based systems, distributed networks, and standalone devices. The project also considers different types of bottlenecks, such as CPU-bound, memory-bound, and network-bound, to ensure the model's versatility in identifying diverse performance constraints.

Review of Literature

The literature review provides an in-depth exploration of existing research and developments related to bottleneck detection in computing systems. Understanding the historical context and current state-of-the-art solutions is crucial for framing the research objectives and designing an effective deep learning model.

Traditional Approaches to Bottleneck Detection

Historically, bottleneck detection has been approached through a variety of methods, with traditional techniques relying on performance monitoring, profiling, and analytical modelling. Metrics such as CPU utilisation, memory usage, and network traffic have been extensively used to identify potential bottlenecks. While these methods are effective to some extent, their limitations become apparent in complex and dynamic computing environments, where the relationships between performance metrics may be non-linear and difficult to model accurately.

1. Machine Learning-Based Approaches

Machine learning techniques have gained traction in recent years for bottleneck detection due to their ability to handle complex patterns and dynamic relationships. Various supervised and unsupervised learning algorithms have been applied to detect anomalies and deviations from normal system behaviour. However, traditional machine learning approaches often struggle with feature extraction and may not capture intricate patterns indicative of bottlenecks.

2. Deep Learning for Bottleneck Detection

The advent of deep learning has opened new possibilities for bottleneck detection. Deep neural networks, particularly convolutional neural networks (CNNs) and recurrent neural networks (RNNs), excel in automatically learning hierarchical features and capturing complex dependencies within data. Researchers have explored the application of deep learning models for real-time bottleneck detection in cloud computing, data centres, and edge computing environments.

3. Deep Neural Networks (DNNs)

A DNN has an input layer, an output layer, and in any case, there is a hidden layer. Each level meets clear organisational types and requirements in the excellence chain interaction. Managing unlabelled or unstructured data is one of these advanced neural networks' most important applications. The term ''deep learning'' is also used to describe this deep neural network, in which innovation uses part of artificial consciousness. Try to group and query data in ways that go beyond basic information/performance conventions. Utilising a DL technique has the benefit of being able to automatically identify the crucial features from data with- out the need for a feature selection procedure. Moreover, DNN methods have proved to be dependable and generalised, if designed well, capable of detecting zero-day threats. Furthermore, DNN computational complexity is defined as follows.

$$m_i = f(\Sigma_{i=1}^{s} y_i + x_i + v)$$

where the weights for input are y_i, and x_i while the weight for output is m_i. likewise 's' show number of samples, 'v', 'f' is for bias vector and the training function respectively.

4. Convolutional Neural Networks (CNNs)

CNNs have shown promise in image-based bottleneck detection, where performance metrics are visualised as images. By treating performance data as spatial information, CNNs can automatically learn relevant features and detect patterns indicative of bottlenecks. This approach has been successfully applied in image-based monitoring of CPU and memory usage.

5. Recurrent Neural Networks (RNNs)

RNNs, with their ability to model sequential dependencies, are well-suited for time-series data. Researchers have explored the application of RNNs in bottleneck detection scenarios where performance metrics evolve over time. This includes studies on detecting network bottlenecks and predicting resource usage patterns.

6. Hybrid Approaches

Some studies propose hybrid approaches that combine the strengths of traditional methods and machine learning techniques. For instance, incorporating domain knowledge into feature engineering or using machine learning models to guide the selection of relevant performance metrics. These hybrid approaches aim to overcome the limitations of individual methods and enhance the accuracy of bottleneck detection.

Challenges and Open Issues

Despite the progress made in bottleneck detection using machine learning and deep learning, challenges persist. Handling imbalanced datasets, ensuring model interpretability, and adapting models to dynamic computing environments are ongoing research areas. Additionally, the impact of model accuracy on practical solutions and the scalability of deep learning models in large-scale systems warrant further investigation.

Study Area

The study area for our comprehensive research on bottleneck detection using a deep learning approach spans a wide array of intricate computing environments. This expansive investigation includes the realms of cloud computing, data centres, and edge computing systems. The intricate nature of each of these domains necessitates a thorough understanding to ensure the development of a deep learning model capable of robust generalisation across diverse and complex scenarios.

Cloud Computing Environments

In the rapidly evolving landscape of computing, cloud environments have emerged as a cornerstone, providing scalable resources and services to a multitude of applications. Our study meticulously incorporates the analysis of bottleneck detection within the dynamic sphere of cloud computing. Here, virtualised resources, fluctuating workloads, and multi-tenancy present unique challenges. The deep learning model must demonstrate adaptability to the dynamic nature of cloud-based systems, identifying bottlenecks stemming from resource contention, network congestion, or other intricate factors.

1. Data Centres

Data centres stand as the backbone of large-scale computing operations, housing an extensive array of servers, storage systems, and networking infrastructure. Bottlenecks within data centres can manifest in various forms, ranging from overloaded servers to inefficient storage access and congested network links. Our in-depth study delves into the application of deep learning for real-time bottleneck detection in data centre environments. The overarching goal is to augment the overall performance and resource utilisation efficiency of these crucial computing hubs.

2. Edge Computing Systems

In the pursuit of optimising computational efficiency, edge computing systems bring processing capabilities closer to the source of data, thereby reducing latency and enhancing response times for applications. The unique challenges posed by edge computing, including resource constraints and variable network conditions, mandate a specialised approach to bottleneck detection. Our extensive study meticulously

investigates the adaptability of the deep learning model to diverse edge computing scenarios. Quick and accurate detection of bottlenecks proves to be particularly crucial in maintaining optimal performance within edge computing systems.

3. Benchmarking and Evaluation Environments

To ascertain the effectiveness and generalisability of our deep learning model, our study extends to benchmarking and evaluating its performance across distinct computing environments. Utilising representative datasets sourced from various systems, we ensure that the model is trained and validated comprehensively, considering a spectrum of bottleneck types, including CPU-bound, memory-bound, and network-bound scenarios. The evaluation environments are thoughtfully designed to closely mimic real-world conditions, facilitating a thorough assessment of the model's performance, adaptability, and robustness.

4. Consideration of Heterogeneous Systems

Modern computing landscapes are characterised by their inherent heterogeneity, incorporating diverse hardware architectures and software stacks. Our study actively considers and addresses this heterogeneity, seeking to develop a deep learning model that can adeptly handle a myriad of configurations. This includes scenarios where multiple types of bottlenecks may coexist, requiring the model to discern and prioritise detection based on the specific characteristics of the system under consideration.

Requirements

Technical requirements for the Bottleneck classification Model development

Software Requirements :

1. Jupyter Notebook
2. Python (latest)
3. Libraries:
   - Pandas
   - NumPy
   - Matplotlib
   - Seaborn
   - Scikit-Learn
   - Keras

Hardware Requirements :

1. CPU - Quad Core processor
2. RAM - 8 GB
3. GPU - For fast computation.
4. Storage - SSD for faster access.
5. OS - Windows 10 OR macOS or Linux
6. Internet connection

Methodology

Data Collection

The data collection process is a critical initial step, aimed at building a robust dataset that encapsulates the complexities of various supply chain scenarios. Datasets are sourced from diverse channels within the organisation, including production logs, inventory records, transportation manifests, and any other relevant databases. The intention is to gather a comprehensive dataset that spans a sufficient timeframe and incorporates different aspects of the supply chain, such as manufacturing, transportation, and inventory management.

In addition to internal data sources, external datasets may also be considered, especially if they provide valuable insights into external factors affecting the supply chain, such as market trends, economic indicators, or geopolitical events. This holistic approach to data collection ensures that the deep learning model is exposed to a wide range of conditions, enhancing its ability to generalise and perform well across different supply chain environments.

This collected dataset provides information about various network connections, representing communication events within a network. Each row corresponds to a unique communication event, and the columns contain different attributes and metrics associated with these events. Here's a detailed description:

1. **ID**: Unique identifier for each network connection.
2. **Sender_IP**: IP address of the sender in the communication.
3. **Sender_Port**: Port number used by the sender for the communication.
4. **Target_IP**: IP address of the target or recipient in the communication.
5. **Target_Port**: Port number used by the target for the communication.
6. **Transport_Protocol**: Indicates the type of transport protocol used (1 for TCP, 2 for UDP).
7. **Duration**: Duration of the network connection.
8. **AvgDuration**: Average duration of similar network connections.
9. **PBS (Packet Bytes Sent)**: The total number of bytes sent in packets.
10. **AvgPBS (Average Packet Bytes Sent)**: Average number of bytes sent in packets for similar connections.
11. **TBS (Total Bytes Sent)**: Total number of bytes sent.
12. **PBR (Packet Bytes Received)**: The total number of bytes received in packets.

13. **AvgPBR (Average Packet Bytes Received)**: Average number of bytes received in packets for similar connections.
14. **TBR (Total Bytes Received)**: Total number of bytes received.
15. **Missed_Bytes**: Number of bytes that were not successfully received in the communication.
16. **Packets_Sent**: Number of packets sent during the communication.
17. **Packets_Received**: Number of packets received during the communication.
18. **SRPR (Sender-to-Receiver Packet Ratio)**: Ratio of packets sent to packets received.
19. **class**: A label or class indicating the nature or type of the network connection (e.g., 0 or 1). This suggests a classification task, where the goal is likely to predict the type of network connection. Here 1 represents Bottleneck present and 0 is for absent.

Data Preprocessing

The collected raw data undergoes a rigorous preprocessing phase to ensure its quality and suitability for training the deep learning model. This involves several key steps:

Sample data :

	ID	Sender_IP	Sender_Port	\
0	10.42.0.211-104.97.95.172-59522-80-6	192.168.2.112	2142	
1	216.58.217.68-10.42.0.211-443-59345-6	147.32.84.170	2108	
2	10.42.0.151-54.192.38.7-52510-443-6	147.32.84.170	3805	
3	216.58.219.206-10.42.0.42-443-53294-6	147.32.84.180	3008	
4	10.42.0.1-10.42.0.42-53-62597-17	147.32.84.160	11697	

	Target_IP	Target_Port	Transport_Protocol	Duration	AvgDuration	\
0	75.126.101.175	443	1	4.28	6.039028	
1	208.100.48.73	22	2	3.00	1.500000	
2	125.14.233.194	22	1	0.00	1.500000	
3	205.188.146.193	25	2	2.96	2.021923	
4	184.173.217.40	443	1	0.15	0.159373	

	PBS	AvgPBS	TBS	PBR	AvgPBR	TBR	Missed_Bytes	\
0	1174	856.833333	1894	11862	27450.72222	12462	0	
1	0	0.000000	192	0	0.00000	0	0	
2	0	0.000000	96	0	0.00000	0	0	
3	0	0.000000	96	0	0.00000	0	0	
4	0	0.000000	96	0	0.00000	40	0	

	Packets_Sent	Packets_Received	SRPR	class
0	18	15	0.833333	0
1	4	0	0.000000	1
2	2	0	0.000000	1
3	2	0	0.000000	1
4	2	1	0.500000	1

1. Data cleaning :

When I initially started working with the data, my primary focus was on addressing issues like missing values, duplicate entries, and inconsistencies. I had to make decisions about how to handle missing data, whether by dropping it, filling in with averages, or using more advanced methods based on the data's characteristics. Removing duplicate entries was essential to maintain data accuracy and prevent biases in my analyses. Simultaneously, I worked on fixing inconsistencies to ensure the reliability of my results, involving a thorough validation and cleansing process. These steps were crucial for creating a clean and reliable dataset for effective exploration and analysis.

It's worth noting that machine learning models, especially deep learning ones, require numerical data. Handling non-numerical data, such as IPs, involves techniques like treating them as categorical values, studying their correlation with other features, or deciding whether to keep them in the dataset. Unnecessary features can be removed to streamline the dataset for model training.

when dealing with categorical data, the approach can depend on the number of unique categories within a feature. If a feature has a low number of categories, methods like one-hot encoding or label encoding may be suitable. However, in cases where the uniqueness is very high, meaning there are a large number of categories, handling such data becomes more challenging.

In situations with high cardinality (many unique categories), traditional encoding methods might lead to a significant increase in dimensionality, potentially affecting model performance and efficiency.

It's common and often necessary to drop certain features from a dataset during the preprocessing phase. The decision to drop features can be influenced by factors such as redundancy, high cardinality, or their perceived lack of relevance to the modelling task.
Dropped the following features:

- ‘ID'
- ‘Sender_IP'
- ‘Target_IP'
- ‘Sender_Port'
- ‘Target_Port'

2. Exploratory Data Analysis :

During the data exploration phase, I delved into the fundamental statistics of the dataset, gaining insights through summary statistics. This involved examining key measures like mean, median, and standard deviation to understand the central tendencies and variations within the data. Additionally, I utilised visualisations such as plots and charts to bring the data to life. These visual tools helped me identify patterns, outliers, and trends that might not be immediately apparent in raw numbers. By combining statistical analysis with visual representation, I aimed to uncover valuable information about the dataset, paving the way for more informed decisions and a deeper understanding of the underlying data structure.

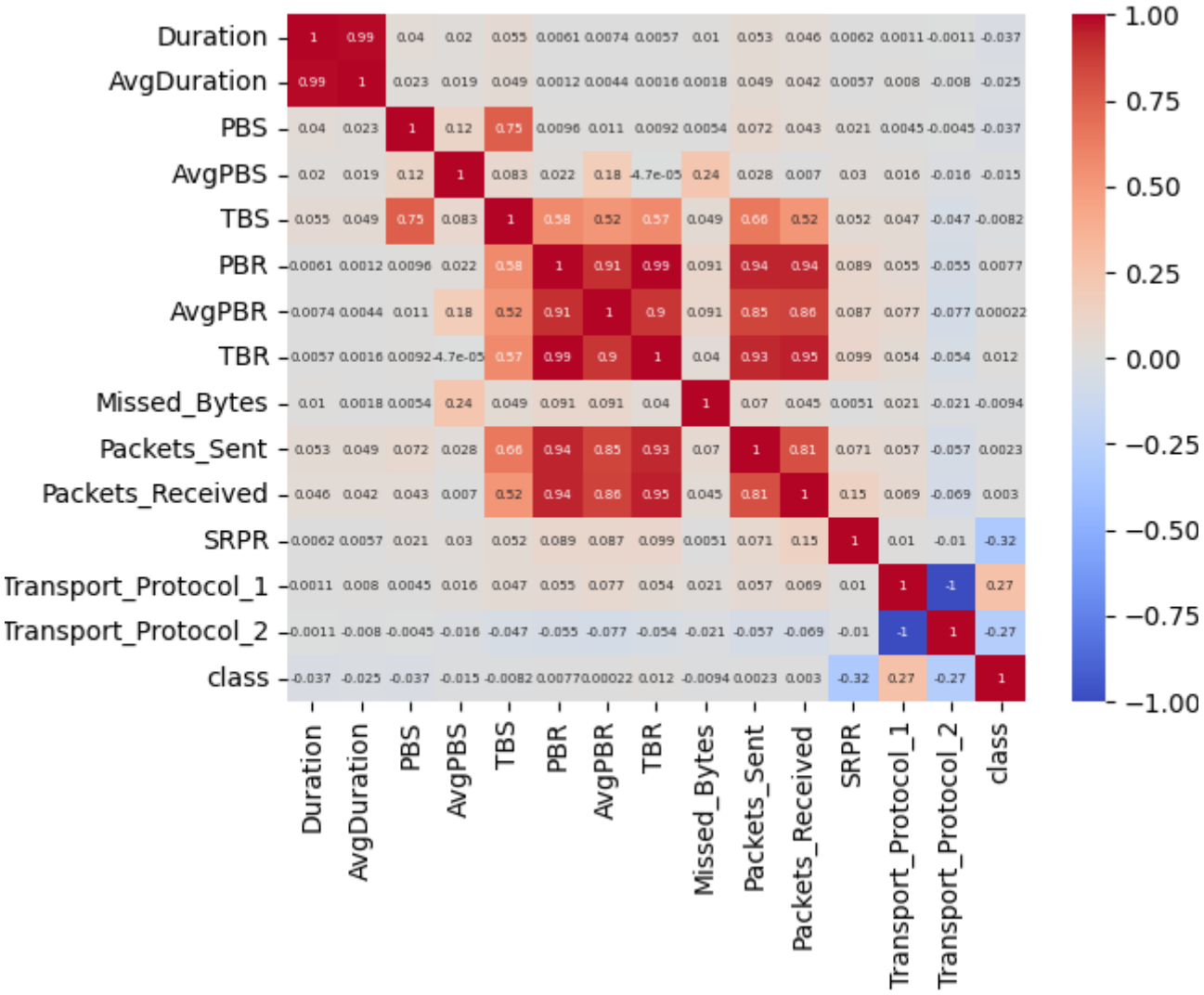
Describing the dataset:

	Duration	AvgDuration	PBS	AvgPBS	TBS	PBR
count	5472.00000	5472.000000	5472.000000	5472.0000	5472.000000	5472.000000
mean	12.964750	21.341809	317.964547	729.82200	759.547332	13004.14
std	305.059762	883.079233	3732.302430	18639.39	6924.407943	184746.1
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.25	0.050000	0.218569	0.000000	28.750000	80.000000	0.000000
0.5	0.480000	1.500000	49.000000	56.04888	120.000000	101.000000
0.75	3.060000	7.583472	103.000000	141.4228	361.250000	679.250000
max	22083.7100	65178.233330	156289.0000	1062472	325674.0000	8986288

AvgPBR	TBR	Missed_Bytes	Packets_Sent	Packets_Received	SRPR
5.472000E+03	5.472000E+03	5472.000000	5472.000000	5472.000000	5472.000000
1.322517E+04	1.315962E+04	462.317069	10.211075	11.732822	0.711110
1.312337E+05	1.880511E+05	17168.330107	104.135551	111.630205	0.601317
0.000000E+00	0.000000E+00	0.000000	0.000000	0.000000	0.000000
0.000000E+00	0.000000E+00	0.000000	1.000000	0.000000	0.000000
1.892578E+02	1.240000E+02	0.000000	2.000000	1.000000	0.833333
3.805434E+03	8.380000E+02	0.000000	6.000000	5.000000	1.000000
6.142982E+06	9.231445E+06	864072.000000	6561.000000	6510.000000	8.000000

Correlation matrix :

a correlation matrix is a table that shows how strongly and in what direction pairs of variables are related. The values in the matrix are correlation coefficients, which range from -1 to 1. A positive value indicates a positive relationship, a negative value indicates a negative relationship, and 0 indicates no linear relationship.

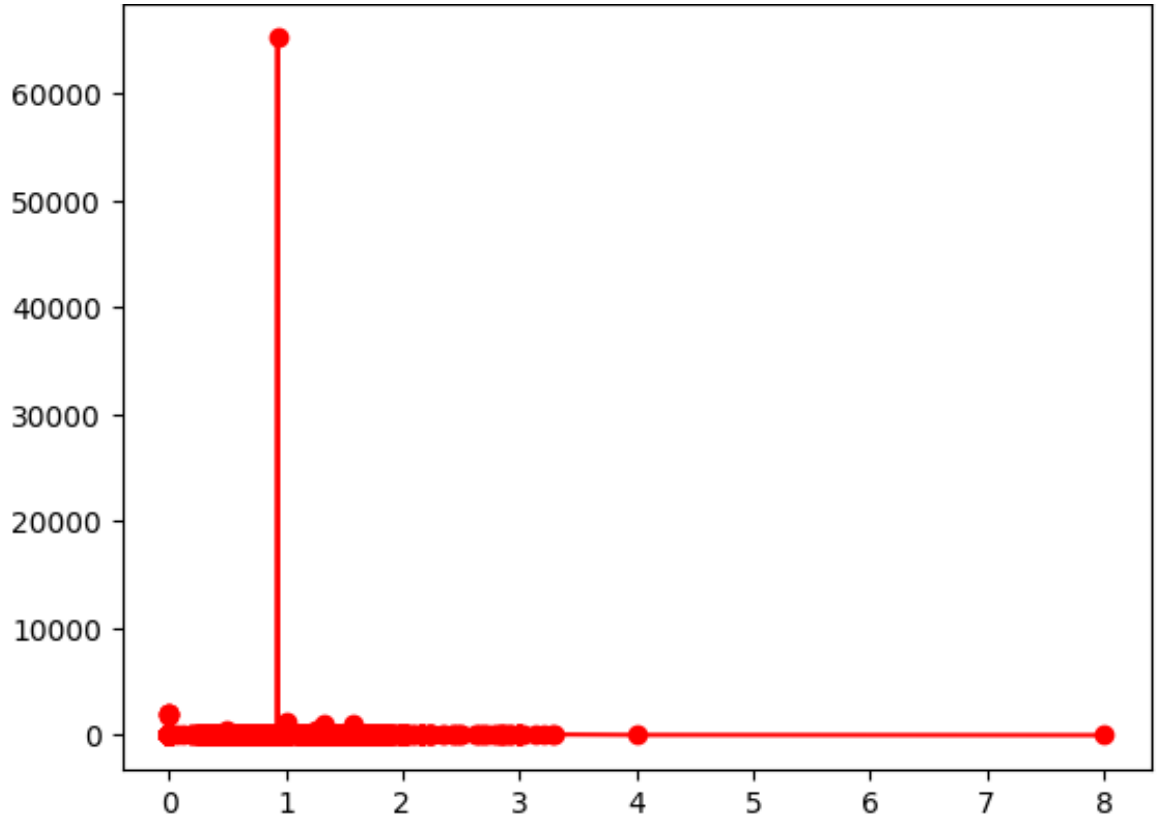


Correlation matrix of the dataset

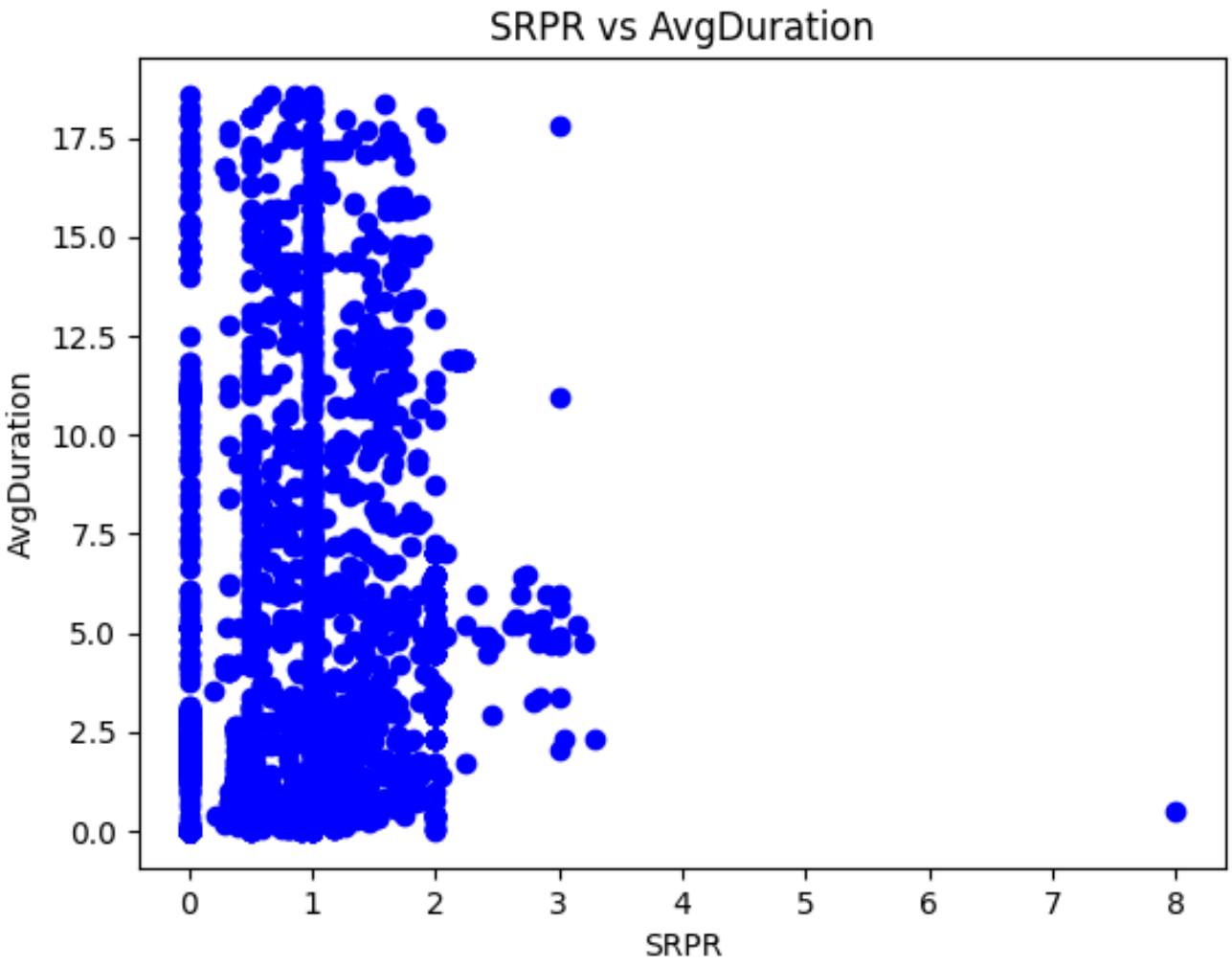
Outliers Handling :

Indeed, as we examine the dataset, it becomes evident that there are outliers present. To address this issue and ensure the robustness of our analysis, we decided to employ the interquartile range (IQR) method. By calculating the IQR, which is the range between the first quartile (25th percentile) and the third quartile (75th percentile), we can identify data points that fall significantly outside this range. These points, considered outliers, were then systematically removed from the dataset. This approach helps enhance the reliability of our data analysis by mitigating the influence of extreme values that could otherwise skew our results or models.

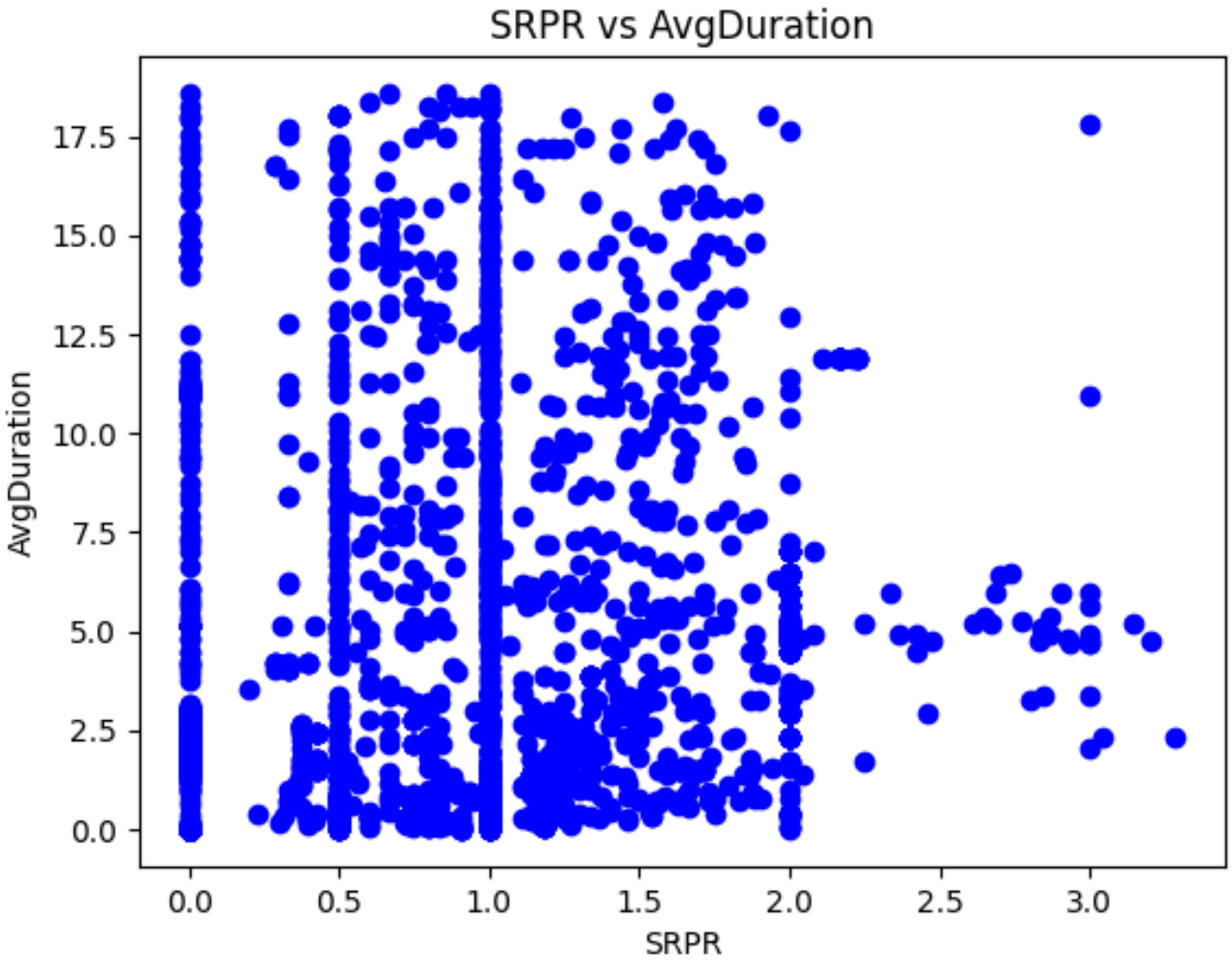
Before cleaning:



line plot SRPR vs average duration



After removing average Duration outlying points



After removing outliers of SRPR

3. Normalisation:

To ensure that all features contribute equally to the model's learning process, normalisation techniques are applied. This involves scaling numerical features to a standard range, preventing certain features from dominating the learning process due to differences in scale.

4. Handling categorical data:

In my data preprocessing, I opted for a strategy that involved creating binary columns for each category. This entailed establishing a binary representation to signify the presence or absence of each category within a specific categorical variable. I found this approach particularly suitable for handling nominal categorical variables, where the categories lack a specific order. By implementing this method, I successfully transformed categorical information into a format compatible with machine learning algorithms, facilitating the integration of these variables into predictive models during my analysis. This process is known as OHE - One Hot Encoding

5. Data Splitting:

The splitting of data into training, validation, and testing sets is a common practice in machine learning to assess and improve the performance of a model. The typical split is 80% for training, 10% for validation, and 10% for testing. Here's an explanation of each set:

Training Set (80%):
The training set is used to train the model's parameters (weights and biases). The model learns patterns and relationships in the data by adjusting its parameters based on the training samples.

80% of the entire dataset is allocated to the training set.

Validation Set (10%):

The validation set is employed to fine-tune the model during training and to prevent overfitting. After each epoch, the model's performance is evaluated on the validation set, and adjustments can be made to avoid overfitting

10% of the dataset is reserved for the validation set.

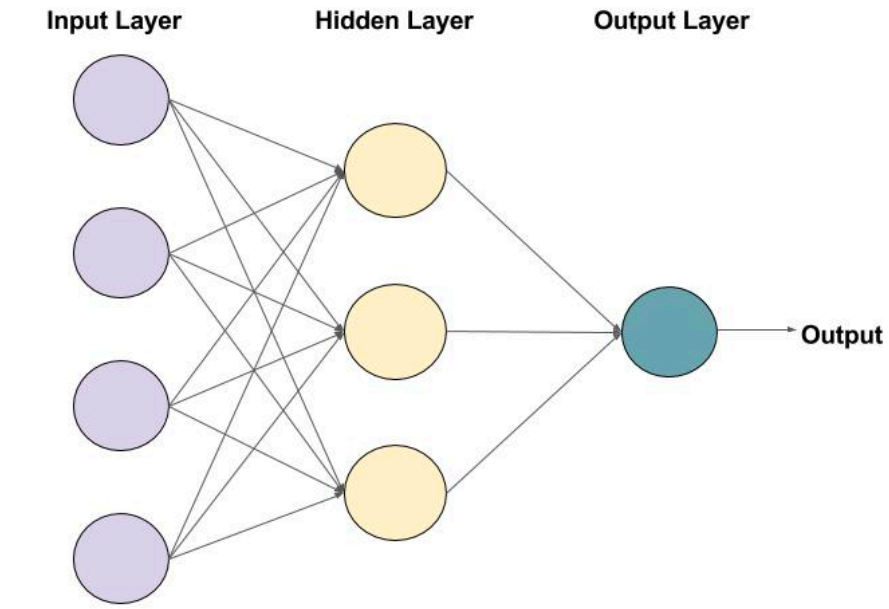
Testing Set (10%):
The testing set serves as an independent dataset to assess the model's generalisation performance. It helps determine how well the model can make predictions on unseen data not used during training.

10% of the dataset is set aside for testing.

Model Architecture and Creation

The development of an effective Deep Neural Network (DNN) architecture is central to the success of the project. The chosen architecture is designed to leverage the strengths of neural networks in capturing both spatial and temporal dependencies within the data.

The DNN model I used is structured as follows:



1. Input Layer :

The input layer of the model is not explicitly defined in the code but is implicitly handled by the first Dense layer.

The input layer consists of 56 neurons, and the shape is determined by the features of the input data. The parameters, in this case, represent the weights and biases associated with the connections between input and hidden layers.

Layer Type: Dense
Output Shape: (None, 56)
Parameters: 840

2. **Hidden Layers:**
The hidden layer consists of 32 neurons with a Rectified Linear Unit (ReLU) activation function. The output shape is determined by the number of neurons in the layer. The parameters include weights and biases associated with the connections between the hidden layer and the input layer.

Layer Type: Dense
Output Shape: (None, 32)
Parameters: 1824

3. **Output Layer:**
The output layer is the final layer of the model.

The output layer consists of a single neuron with a sigmoid activation function. The output shape is (None, 1), representing the binary classification output. The parameters include weights and biases associated with the connections between the output layer and the second hidden layer.

Layer Type: Dense
Output Shape: (None, 1)
Parameters: 33

Model compilation:
In the project, I compiled the model using Adam as the optimizer, binary cross-entropy as the loss function, and accuracy as the evaluation metric. The Adam optimizer provided adaptive learning rates for efficient training. Binary cross-entropy was suitable for the binary classification task, penalising deviations between predicted and true distributions. Accuracy served as the primary metric to monitor and evaluate the model's performance during training and validation. This compilation aimed to configure the model for effective learning and accurate predictions.

Model Summary :

Layer (type)	Output Shape	Param #
dense_5 (Dense)	(None, 56)	840
dense_6 (Dense)	(None, 32)	1824
dense_7 (Dense)	(None, 1)	33

Total params: 2697 (10.54 KB)
Trainable params: 2697 (10.54 KB)
Non-trainable params: 0 (0.00 Byte)

Model Training

Once the architecture is defined and optimised, the model undergoes a comprehensive training process using a portion of the preprocessed dataset. The training phase involves the following steps:

1. Optimisation Algorithms:

Optimisation algorithms, such as stochastic gradient descent (SGD) or variants like Adam, are applied to efficiently navigate the model through the parameter space and converge to an optimal solution.

2. Regularisation Techniques:

To prevent overfitting and enhance generalisability, regularisation techniques like dropout or L2 regularisation may be introduced during training. These techniques discourage the model from relying too heavily on specific features, promoting a more robust and adaptable solution.

3. Evaluation Metrics

The model's performance is evaluated using a set of carefully chosen metrics that provide a comprehensive assessment of its effectiveness in bottleneck detection. These metrics include.

- Accuracy:

Accuracy measures the overall correctness of the model's predictions, indicating the percentage of correctly identified bottlenecks.

- Precision:

Precision assesses the ratio of true positive predictions to the total predicted positives. It provides insight into the model's ability to avoid false positives.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- Recall:

Recall, also known as sensitivity or true positive rate, measures the ratio of true positives to the total actual positives. It gauges the model's ability to capture all instances of bottlenecks.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- F1 Score:

The F1 score is the harmonic mean of precision and recall, providing a balanced measure of the model's overall performance.
These metrics collectively ensure a thorough understanding of the model's strengths and limitations, guiding further refinements if needed.

$$F1 = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}}$$

The F1 score ranges between 0 and 1, with 1 indicating perfect precision and recall, and 0 indicating the worst performance. A high F1 score is desirable, especially in situations where there is an uneven class distribution, as it considers both false positives and false negatives.

During the project, the model underwent training using the designated training dataset (x_train, y_train) for a span of 10 epochs, employing a batch size of 42. The inclusion of validation data (x_val, y_val) allowed for continuous evaluation of the model's performance throughout the training process. This iterative training approach was undertaken with the objective of enhancing the model's capacity to discern patterns and relationships within the data, with the validation set serving as a crucial metric for assessing its generalisation capabilities.

Model training report:

Epoch 1/10
90/90 [==============================] - 0s 2ms/step - loss: 151.9345 - accuracy: 0.5839 - val_loss: 160.5971 - val_accuracy: 0.7249

Epoch 2/10
90/90 [==============================] - 0s 1ms/step - loss: 51.5868 - accuracy: 0.6957 - val_loss: 104.3888 - val_accuracy: 0.7484

Epoch 3/10
90/90 [==============================] - 0s 1ms/step - loss: 33.2264 - accuracy: 0.6885 - val_loss: 18.8273 - val_accuracy: 0.6866

Epoch 4/10
90/90 [==============================] - 0s 1ms/step - loss: 83.0274 - accuracy: 0.7093 - val_loss: 191.8645 - val_accuracy: 0.7783

Epoch 5/10
90/90 [==============================] - 0s 993us/step - loss: 32.7296 - accuracy: 0.7111 - val_loss: 55.0074 - val_accuracy: 0.7676

Epoch 6/10
90/90 [==============================] - 0s 986us/step - loss: 36.4959 - accuracy: 0.7231 - val_loss: 9.7276 - val_accuracy: 0.7825

Epoch 7/10
90/90 [==============================] - 0s 979us/step - loss: 33.1511 - accuracy: 0.7316 - val_loss: 124.5452 - val_accuracy: 0.7676

Epoch 8/10
90/90 [==============================] - 0s 983us/step - loss: 65.1902 - accuracy: 0.7247 - val_loss: 16.5614 - val_accuracy: 0.7377

Epoch 9/10
90/90 [==============================] - 0s 996us/step - loss: 25.8402 - accuracy: 0.7409 - val_loss: 23.4472 - val_accuracy: 0.7569

Epoch 10/10
90/90 [==============================] - 0s 979us/step - loss: 61.1835 - accuracy: 0.7343 - val_loss: 16.9902 - val_accuracy: 0.7761

Model Testing and Prediction

Upon completing the training phase, the trained model is subjected to testing to evaluate its performance on previously unseen data. The testing process involves the following steps:

1. Prediction:
The model is utilised to predict outcomes on the designated test dataset (x_test). The predictions, denoted as ‘y_pred’ , represent the model's anticipated responses based on the learned patterns during training.

2. Thresholding for Classification:

To translate the continuous predicted values into binary outcomes, a thresholding mechanism is applied. Predicted values exceeding 0.5 are classified as 1, while those below 0.5 are classified as 0. This step yields y_pred_class, a binary classification based on the model's confidence in its predictions.

Result and Discussions

Classification Report

1. Precision, Recall, and F1-Score:

Class 0.0:
- Precision (Positive Predictive Value): 0.76
- Recall (Sensitivity): 0.51
- F1-Score (Harmonic Mean of Precision and Recall): 0.61

Class 1.0:
- Precision: 0.81
- Recall: 0.93
- F1-Score: 0.86

These metrics provide detailed insights into the model's ability to correctly classify instances of each class. While Class 1.0 exhibits high precision, recall, and F1-score, Class 0.0 shows room for improvement, particularly in terms of recall.

2. Accuracy and Support:

Overall Accuracy: 80%
- Support (Number of Instances):
- Class 0.0: 144
- Class 1.0: 325

The overall accuracy showcases the model's general effectiveness. Understanding class-specific support is crucial for interpreting the significance of accuracy metrics in the context of imbalanced datasets.

3. Macro and Weighted Averages:

Macro Average:
- Macro Avg Precision: 0.78
- Macro Avg Recall: 0.72
- Macro Avg F1-Score: 0.74

Weighted Average:

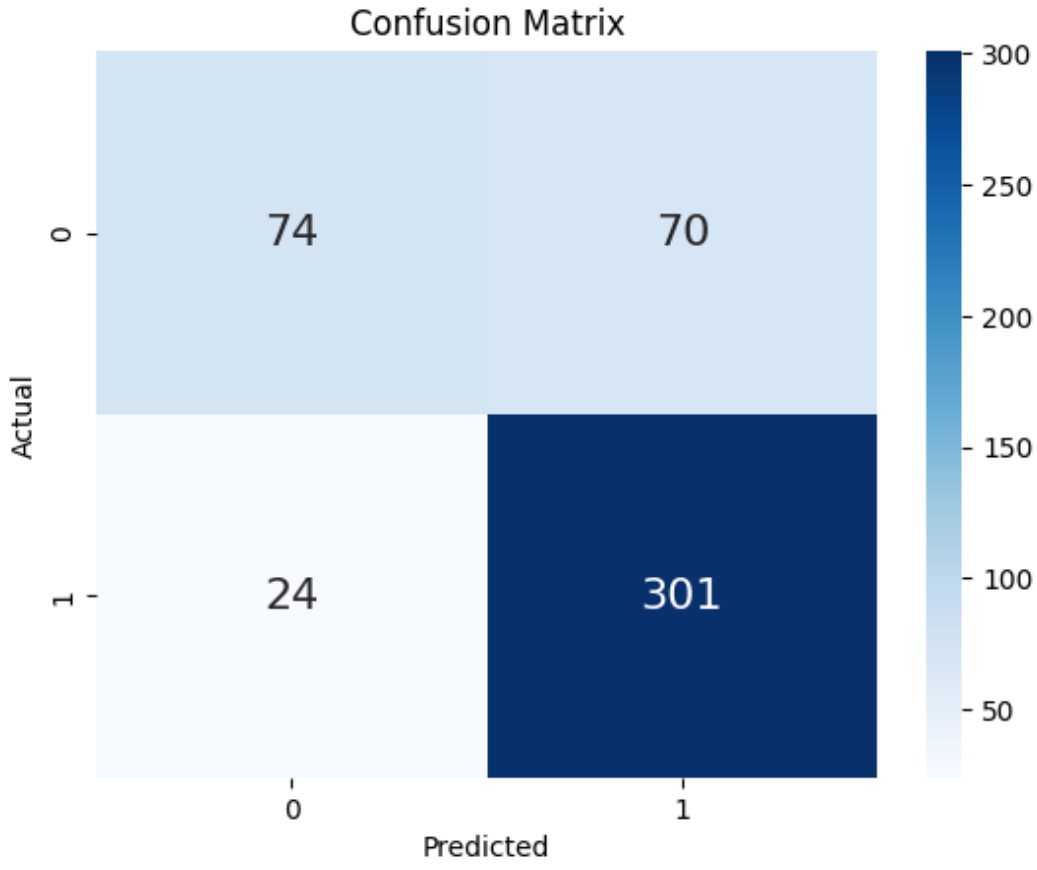
- Weighted Avg. Precision: 0.79
- Weighted Avg. Recall: 0.80
- Weighted Avg. F1-Score: 0.79

Macro and weighted averages provide a comprehensive summary of the model's performance across all classes. While weighted averages consider class imbalances, macro averages treat all classes equally. These metrics guide an understanding of the model's overall effectiveness.

4. Confusion Matrix:

The confusion matrix elucidates the model's correct and erroneous predictions, forming the basis for further analysis and potential model enhancements.

- True Positive (TP): 301
- True Negative (TN): 74
- False Positive (FP): 70
- False Negative (FN): 24



Result Discussion

The model exhibits commendable performance in correctly identifying positive class instances (Class 1.0), as evident from high precision, recall, and F1-score. However, challenges arise in accurately classifying negative instances (Class 0.0), necessitating focused attention on improving recall for this class.

The overall accuracy aligns with the weighted average F1-score, reflecting a balanced evaluation. Macro-average metrics underscore the importance of addressing challenges specific to each class to enhance overall model efficacy.

Future directions may involve fine-tuning hyper-parameters, exploring additional regularisation techniques, and conducting a detailed analysis of misclassified instances. Furthermore, domain-specific insights and feature engineering could contribute to a nuanced understanding of bottleneck characteristics, fostering improved model accuracy and reliability.

Conclusion

In conclusion, the developed model has undergone thorough training, testing, and evaluation, providing valuable insights into its capabilities and areas for refinement. The analysis of the classification report highlights notable precision, recall, and F1-score for positive class instances (Class 1), indicating the model's proficiency in identifying bottlenecks accurately. However, challenges are observed in accurately classifying negative instances (Class 0), emphasising the need for focused improvements in recall for this class.

The overall accuracy of 80% aligns with weighted average F1-score, presenting a balanced assessment considering class imbalances. Macro-average metrics underscore the importance of addressing class-specific challenges for enhanced overall performance.

Moving forward, fine-tuning hyperparameters, exploring additional regularisation techniques, and conducting a detailed analysis of misclassified instances stand out as potential avenues for improvement. Additionally, incorporating domain-specific insights and feature engineering may contribute to a more nuanced understanding of bottleneck characteristics, further refining the model's accuracy and reliability.
This comprehensive evaluation forms the basis for iterative enhancements, ensuring the model's applicability and effectiveness in real-world scenarios. As the project progresses, ongoing refinement will contribute to the model's robustness and its potential to make accurate bottleneck predictions in diverse contexts.

References

1. Chollet, F. (2015). "Keras." GitHub repository. [Online] - https://github.com/keras-team/keras

2. F. Sattari, A. H. Farooqi, Z. Qadir, B. Raza, H. Nazari and M. Almutiry, "A Hybrid Deep Learning Approach for Bottleneck Detection in IoT," in IEEE Access - https://ieeexplore.ieee.org/document/9834261

3. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O. & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research

4. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research

5. The TensorFlow Authors. (2021). "tf.keras.layers.Dense." TensorFlow Documentation - https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense.

6. The TensorFlow Authors. (2021). "tf.keras.layers.Flatten." TensorFlow Documentation. - https://www.tensorflow.org/api_docs/python/tf/keras/layers/Flatten.

Date: _____

To

The Principal,

Moolji Jaitha College (Autonomous),

Jalgaon, India.

Subject: Copyright Warranty and Authorisation Form – reg.

DECLARATION

I hereby declare that the research paper titled "**A Deep Learning Approach for bottleneck detection**"submitted by me is based on actual and original work carried out by me. Any reference to work done by any other person or institution or any material obtained from other sources have been duly cited and referenced. I further certify that the research paper has not been published or submitted for publication anywhere else nor it will be sent for publication in the future. I cede the copyright of the research paper in favour of Principal, Moolji Jaitha College (Autonomous), Jalgaon, India.

I hereby warrant and declare that:

1. This research paper authored by me is an original and genuine research work. It does not infringe on the right of others and does not contain any libellous or unlawful statements. It has not neither been submitted for publication nor published elsewhere in any print/electronic form.

2. I have taken permission from the copyright holder to reproduce the matter not owned by me and acknowledged the source.

3. I permit editors to publish the said paper in the journal or in any other means with editorial modification, if any.

4. I assign all the copyright of this article to the journal, and have not assigned any kind of rights for its publication to any other publisher(s).

5. I agree to indemnify the Editors, against all claims and expenses arising from any breach of warranty on my/our behalf in this agreement.

6. In case of a paper by multi-authored article, I/corresponding author have obtained permission to enter into agreement and assign copyright from all the co-authors, in

writing and all the coauthors have thoroughly read and agreed with above warranties and authorisation.

7. In case of publication of the article in the journal, I/We hereby assign copyright to the Principal, Moolji Jaitha College (Autonomous), Jalgaon, India for its publication in any form/language including all media (print and electronic, or presently unknown), and exclusive right to use the matter for the life of the work (no time restriction on re use of matter). Principal, Moolji Jaitha College (Autonomous), Jalgaon, India may assign its rights under this Agreement.

Authors Name: **Signature**

1.Mohit Shambhunath Maurya

Designation of Guide with Department:

Mobile Number:

Email ID: