Shweta Rana

Network traffic classifier

Introduction:

A Network Traffic Classifier (NTC) is a system or method used to categorize or classify network traffic based on various attributes, patterns, or features. Its primary purpose is to analyze and identify different types of network traffic flowing through a network, such as Internet of Things (IoT) devices, applications, protocols, or services. The goal of NTC is to understand and distinguish between different traffic types, which can be beneficial for various purposes, including network management, security, Quality of Service (QoS) enforcement, and traffic optimization.

Types of Network Traffic Classifiers:

1. Signature-Based Classifiers: These classifiers use predefined patterns or signatures of known network traffic types to recognize and classify them. They are effective for identifying well-known protocols and services but might struggle with new or unknown traffic types.

2. Statistical-Based Classifiers: Statistical classifiers analyze network traffic based on statistical characteristics like packet size, inter-arrival time, or payload distribution. These classifiers can detect anomalies and unknown traffic patterns but might require more processing power and data to achieve accurate results.

3. Machine Learning-Based Classifiers: These classifiers leverage machine learning algorithms to automatically learn and recognize patterns in network traffic. They can handle new and emerging traffic types, making them more adaptable. Examples of machine learning algorithms used in NTC include Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), as mentioned in the research paper's title.

NTC is a critical component in this new scenario[1][2],allowing to detect the services used by dissimilar devices with very different user-profile. The Need for Network Traffic Classification in the IoT Research Paper is that Internet of Things has seen significant growth, with various IoT devices generating diverse network traffic patterns. Traditional methods of traffic classification may not be well-suited to handle the unique characteristics of IoT traffic. There are several approaches to NTC: port-based, payload based, and flow statistics-based [3][4].

The need for this specific method arises due to the following reasons:

1. IoT Traffic Heterogeneity: IoT devices can have different communication patterns, data formats, and protocols, making it challenging to classify them using traditional methods.

2. Dynamic IoT Landscape: The IoT ecosystem is constantly evolving with new devices and services being introduced regularly. An adaptive classifier using machine learning techniques like CNNs and RNNs can better handle new and unknown traffic types.

3. Security and QoS Requirements: As IoT devices become more prevalent and critical in various domains, it becomes crucial to ensure security measures and enforce appropriate Quality of Service. NTC helps in identifying potential security threats and prioritizing traffic for QoS management.

4. Efficient Network Management: Understanding the types of IoT traffic can assist network administrators in optimizing the network's performance and resource allocation.

5. Big Data Challenges: With the vast amount of data generated by IoT devices, traditional manual classification becomes impractical. Machine learning-based NTC can handle big data and automate the classification process.

By combining convolutional neural networks (CNNs)[5] and recurrent neural networks (RNNs)[6], the proposed NTC method likely aims to extract both spatial and temporal features from network traffic data. CNNs are good at capturing spatial patterns, while RNNs can model temporal dependencies, making them well-suited for time-series data like network traffic.

In summary, the goal is to develop a specialized NTC system for IoT traffic using advanced machine learning techniques to address the unique challenges and requirements posed by the Internet of Things.

Brief Description:

With the rapid expansion of the Internet of Things (IoT), more and more devices are interconnected, generating massive amounts of network traffic. To keep IoT networks running smoothly, we need a way to understand and manage this traffic efficiently. Network Traffic Classification (NTC) is the key to achieving this goal. It involves figuring out the type of network service being used in a communication flow, like identifying if it's for web browsing (HTTP) or voice calls (SIP).

Here we used a new and improved method for NTC, specifically designed for IoT traffic. Our approach combines two powerful types of neural networks: Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). These neural networks work together to provide better and more accurate results. The best part is that we don't need to manually select complicated features for our method to work well, making it easier to use for different IoT scenarios.

We explore different ways of combining CNNs and RNNs and also look into how the length of the network flow and chosen features affect our results. By effectively managing IoT network traffic and understanding the behavior of various IoT devices and services, NTC technique contributes to a smoother and more efficient IoT experience.

1. Study of AI/ML Classifier:

In recent years, the exponential growth of networked devices has led to an overwhelming volume of network traffic, especially in the context of the Internet of Things (IoT). Effectively managing and securing this diverse and dynamic traffic has become a significant challenge. Network Traffic Classification (NTC) plays a crucial role in addressing these challenges by categorizing network traffic based on various attributes and patterns. Traditional methods for NTC, such as signature-based and statistical approaches, have shown limitations in handling the heterogeneity and complexity of IoT traffic. To tackle these challenges and harness the full potential of traffic classification, researchers have turned to the power of Artificial Intelligence and Machine Learning (AI/ML) techniques. AI/ML classifiers have emerged as promising solutions due to their ability to automatically learn and adapt from data, making them well-suited for the ever-changing landscape of network traffic.

2. AI/ML Classifier:

AI/ML classifiers, also known as machine learning classifiers, are algorithms that can learn patterns and relationships in data and make predictions or decisions based on this learned knowledge. These classifiers can be trained on labelled datasets, where each data point is associated with a known category or class. Once trained, the classifier can predict the category of unseen data based on the patterns it has learned during training. One of the most significant advantages of AI/ML classifiers is their ability to handle complex and high-dimensional data. They can capture intricate patterns that might be challenging to discern using traditional rule-based approaches. Moreover, AI/ML classifiers have shown excellent generalization capabilities, enabling them to classify new and previously unseen data accurately.

a) Neural Network:

Neural networks are a class of AI/ML models inspired by the human brain's structure and function. They consist of interconnected layers of artificial neurons that process and transform data through weighted connections. Neural networks can automatically learn representations of data, making them highly effective in various tasks, including computer vision, natural language processing, and, crucially, network traffic classification.

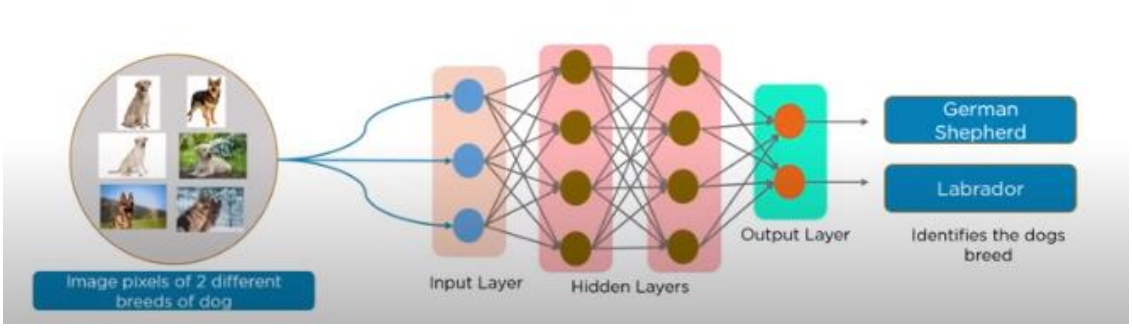


Fig 1: Neural Network

There are several popular types of neural networks, each designed for specific purposes:

- **Feed-Forward Neural Networks:**
  - Feed-Forward Neural Networks are the simplest form of neural networks, where data flows in one direction, from the input layer through one or more hidden layers to the output layer. Each neuron in one layer is connected to every neuron in the subsequent layer. The network learns by adjusting the weights of these connections during the training process to minimize the prediction error.
  - Aim: The primary aim of Feed-Forward Neural Networks is to model and learn complex relationships between input features and corresponding target outputs in a supervised learning setting.
  - How it works: FFNNs pass information through multiple layers, from the input layer to one or more hidden layers and finally to the output layer. The connections between neurons have no cycles or feedback loops, meaning information flows in one direction (forward) through the network.
- **Convolutional Neural Networks (CNNs):**
  - CNNs are a specialized type of neural network designed to process grid-like data, such as images. They are particularly effective in image recognition and computer vision tasks. CNNs use convolutional layers to automatically and adaptively learn spatial hierarchies of features from the input data. The network learns to detect simple features in lower layers (e.g., edges and corners) and progressively more complex patterns in deeper layers.
  - Aim: CNNs are specifically designed for image recognition tasks and spatial data analysis, but they have also been adapted for sequence-based data, such as time-series, including network traffic.
  - How it works: The main aim of CNNs is to efficiently learn and extract spatial patterns and hierarchical features from the input data. This is achieved through convolutional layers, which use filters to detect local patterns, and pooling layers, which reduce spatial dimensions while retaining important information.
- **Recurrent Neural Networks (RNNs):**
  - RNNs are designed to handle sequential data, where the order of the input elements matters. They have loops within their architecture, allowing them to maintain hidden states, which enables them to capture temporal dependencies and context in the data. RNNs are widely used in natural language processing, speech recognition, and time series analysis.
  - Aim: The primary aim of RNNs is to model sequential data and handle temporal dependencies present in time-series data like network traffic.
  - How it works: RNNs have feedback connections, allowing information to be passed from one time step to another. This enables them to capture temporal information and process sequences of varying lengths, making them suitable for tasks like language modeling, speech recognition, and time-series analysis.

- **Deep Neural Networks (DNNs):**
  - Deep Neural Networks refer to neural networks with multiple hidden layers. As neural networks with more layers can learn increasingly complex representations, DNNs are capable of handling intricate patterns and solving highly challenging tasks. Deep learning, which heavily relies on DNNs, has revolutionized many fields in artificial intelligence and machine learning.
  - Aim: The aim of Deep Neural Networks is to leverage the power of multiple hidden layers to learn hierarchical representations of data, enabling the extraction of more abstract and complex features.
  - How it works: DNNs can encompass various architectures, including FFNNs, CNNs, and RNNs, with multiple hidden layers. The main objective is to efficiently represent data at different levels of abstraction, allowing for more accurate and sophisticated learning in a hierarchical manner.

i. **Feed Forward Neural Network:** In it information flows only in forward direction, from the input nodes, though the hidden layer (if any) and to the output nodes there are no cycle or loops in the network.

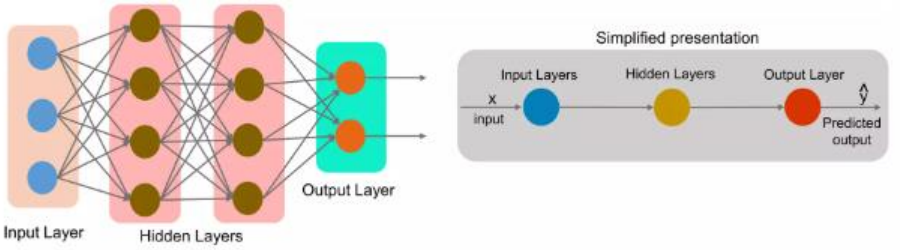


Fig 2: Feed Forward Neural Network

In it:

- Decision are based on current input
- No memory about the past
- No future scope.

Issues in it are:

- Cannot handle sequential data
- Considers only the current input
- Cannot memorise previous inputs.

Solution to it is **RNN:**

- Can handle sequential data
- Considers the current input and also the previous received inputs
- Can memorise previous inputs due to its internal memory

ii. **<u>Recurrent Neural Network</u>** works on the principle of saving the output of layer and feeding this back to the input in order to predict the output of the layer.

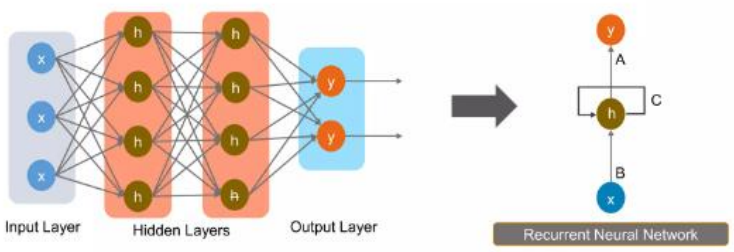


Fig 3: RNN

RNNs come in various types, each catering to specific data structures:

- **One-to-One Network:** Also known as Vanilla Neural Network, it is used for regular machine learning problems without sequential data.
- **One-to-Many Network:** This type generates a sequence of outputs, making it suitable for tasks like image captioning.
- **Many-to-One Network:** It takes a sequence of inputs and produces a single output, often employed in sentiment analysis.
- **Many-to-Many Network:** Utilized for tasks like machine translation, it takes a sequence of inputs and generates a corresponding sequence of outputs.

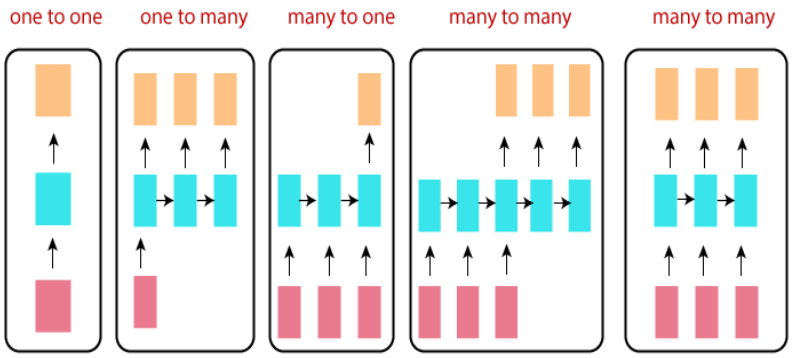


Fig 4: Types of RNN

However, RNNs face challenges during training, such as the vanishing gradient problem, where the gradients become too small, leading to slow training and poor performance, or the exploding gradient problem, where the gradients grow exponentially, causing instability during training.

Solution to Gradient problem:

Fig 5: Solution to Gradient problem

To address these issues, Long Short-Term Memory Networks (LSTM)[7] were introduced. LSTM is a special kind of RNN designed to capture long-term dependencies and retain information for extended periods.

The LSTM network operates through a three-step process:

- Forget irrelevant parts of the previous state.
- Selectively update cell state values.
- Output certain parts of the cell state to produce the final output.

Working of LSTM:

Step1: Decide how much of the part it should remember

In this LSTM decides which information is to be omitted from the cell in that particular time step.

Step2: Decide how much should this unit add to the current state

Step 3: Decide what part of the current cell state makes it to the output.

iii. Convolutional Neural Network:

Used for image recognition:

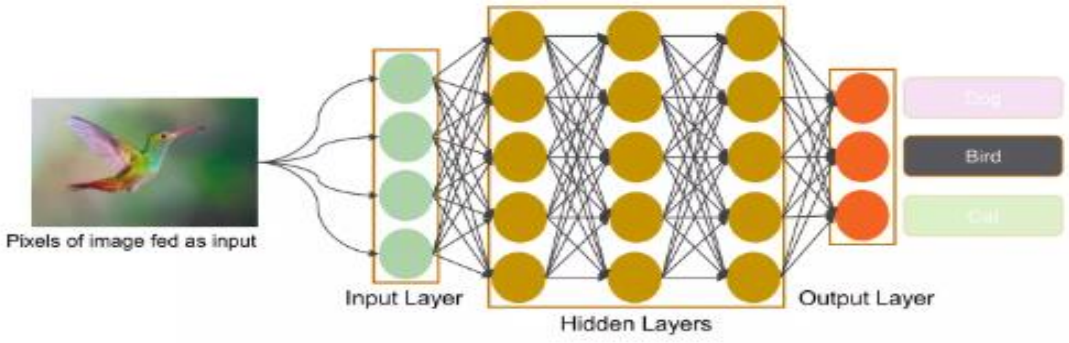


Fig 6: CNN

- *Input layer* accepts the pixels of the image as input in the form of arrays
- *Middle layer* carry out feature extraction by performing certain calculation and manipulation

There are multiple layer like convolution layer, pooling layer etc. that perform feature extraction from the image.

  - **Convolutional layer:** This layer uses a matrix filter and perform convolution operation to detect pattern in the image.
  - **ReLU layer:** ReLU activation function is applied to the convolution layer to get a rectified feature map of the image.
  - **Pooling Layer:** uses multiple filter to detect edges, corners,etc
- *Output Layer:* Finally their is **fully connected layer** that identifies the object in the image.

CNN is a feed forward neural network that is generally used to analyze visual images by processing data with grid like topology.

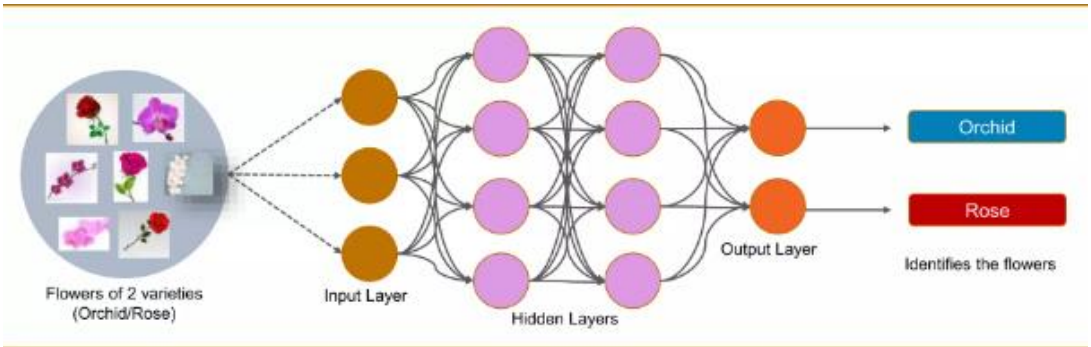
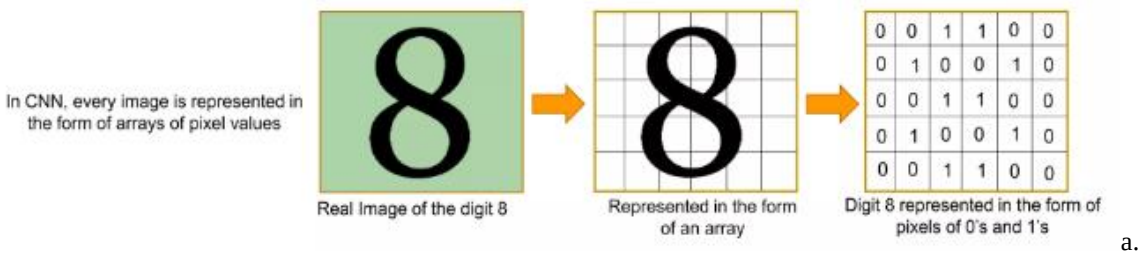
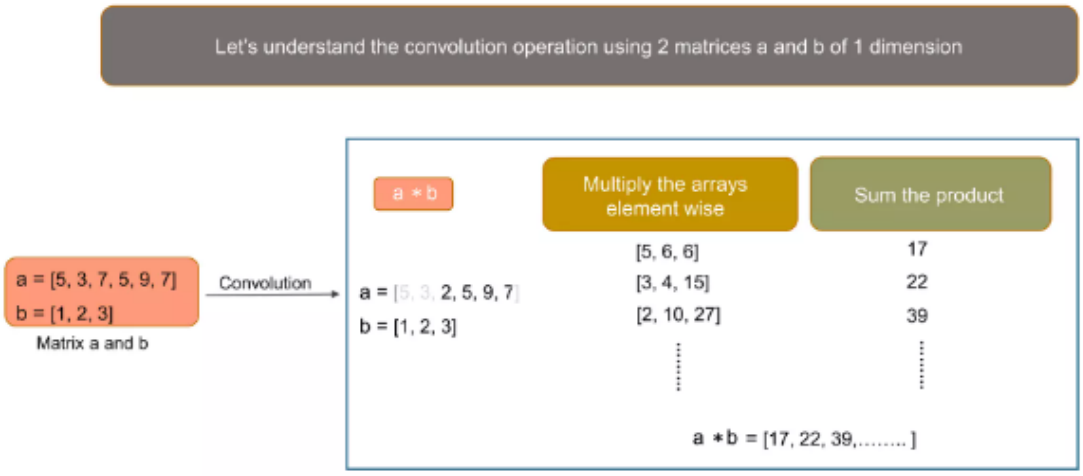


Fig 7: Layers in CNN

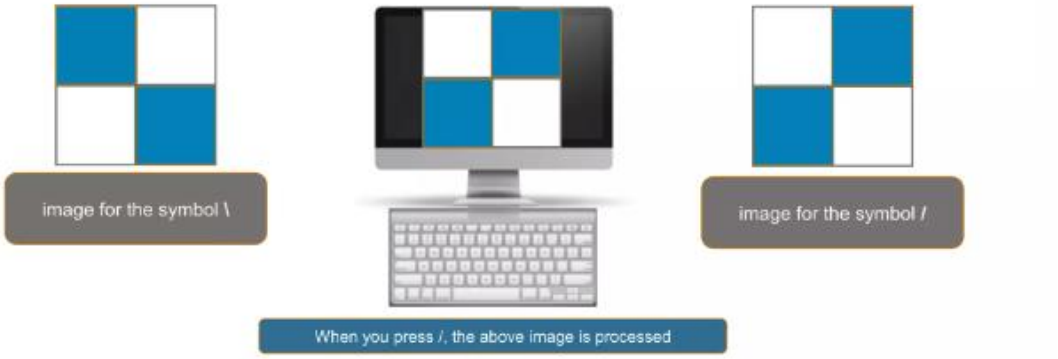
Convolution operation forms the basis of any Convolution Neural Network



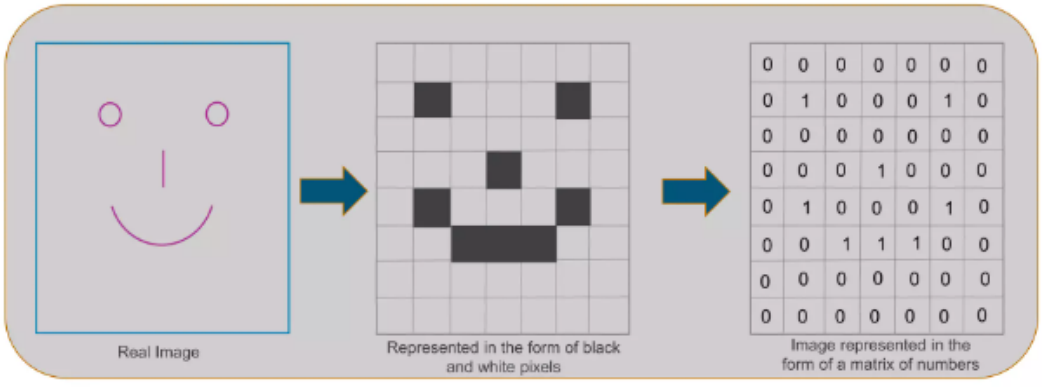
a.



b.



c.



d.

Fig 8: Operations in Convolutional Layer in CNN

Layers of CNN:

- Convolution Layer
- ReLU layer
- Pooling Layer
- Fully Connected Layer

Convolutional Layer: It has a number of filters that perform convolution operation

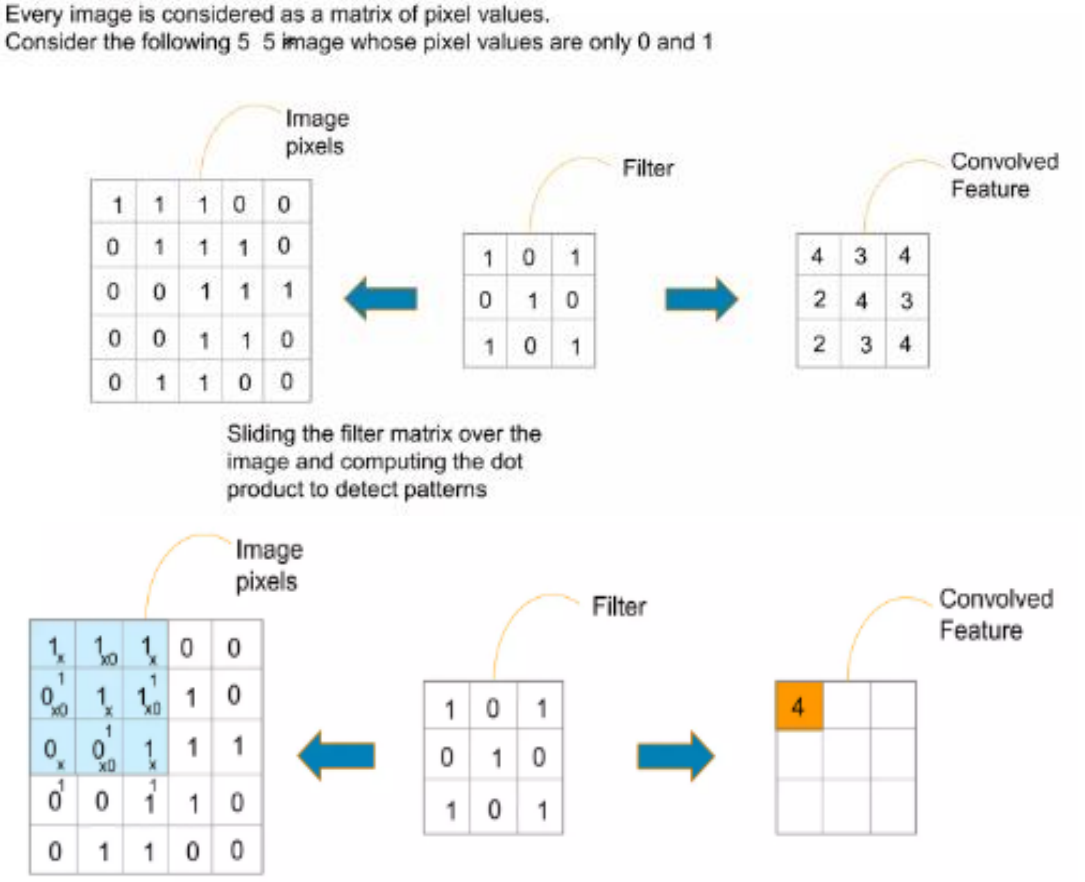


Fig 9: Convolutional Layer

ReLU Layer: Once the feature maps are extracted, the next step is to move them to a ReLU Layer.

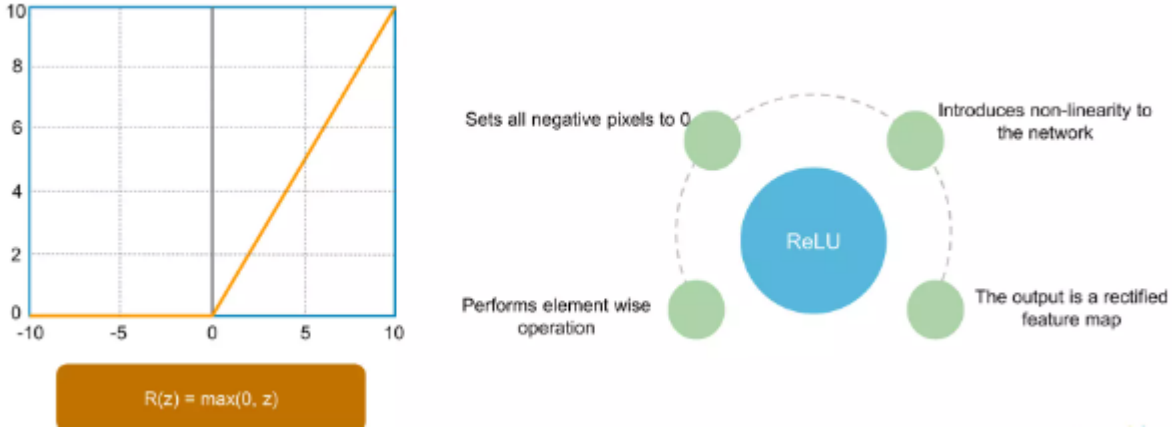


Fig 10: ReLU Layer

Real image is scanned multiple convolution and ReLU layers for locating the features.

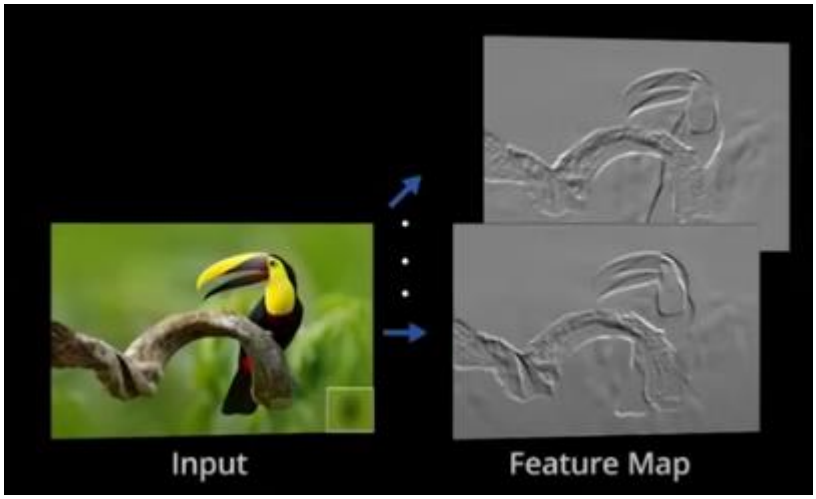


Fig 11: Feature Map in ReLU Layer

There are multiple convolution, ReLU , pooling layer connected one after another that carry out feature extraction.

Pooling Layer: the rectified image map now goes through a pooling layer. Pooling is a don-sampling operation that reduces the dimension of the feature map.

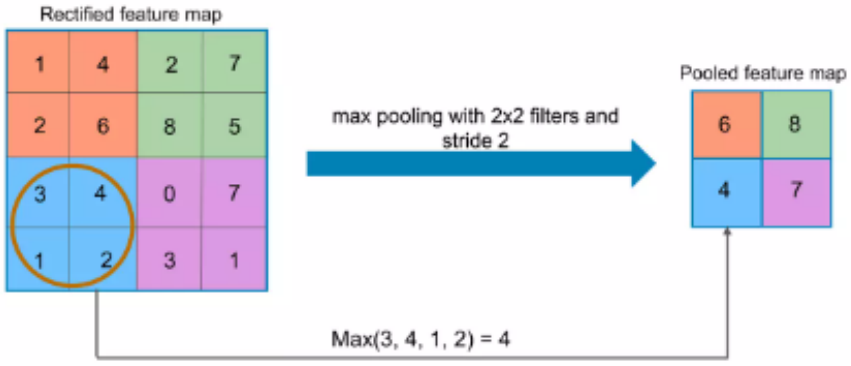


Fig 12: Pooling Layer

Poling layer uses different filters to identify different parts of the image. It identifies edges, corners and other features of the input image.

Flattening: it is a process of converting all the resultant 2D array from pooled feature map into a single long continuous linear vector.

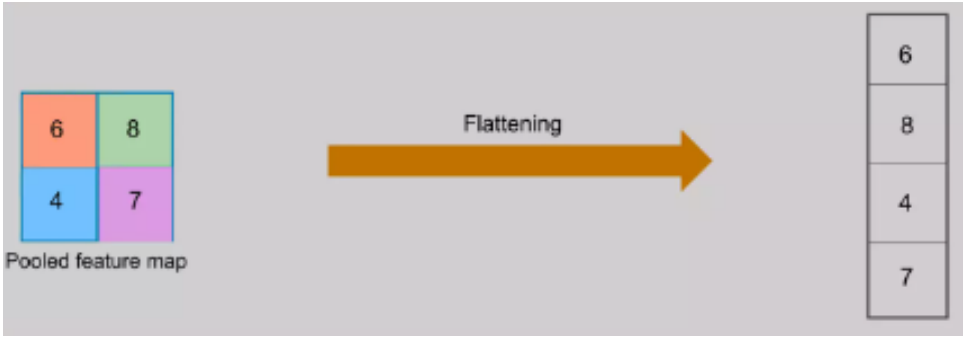


Fig 13: Flattening Layer

Fully Connected Layer: the flattened matrix from the pooling layer is fed as input to the fully connected layer to classify the image

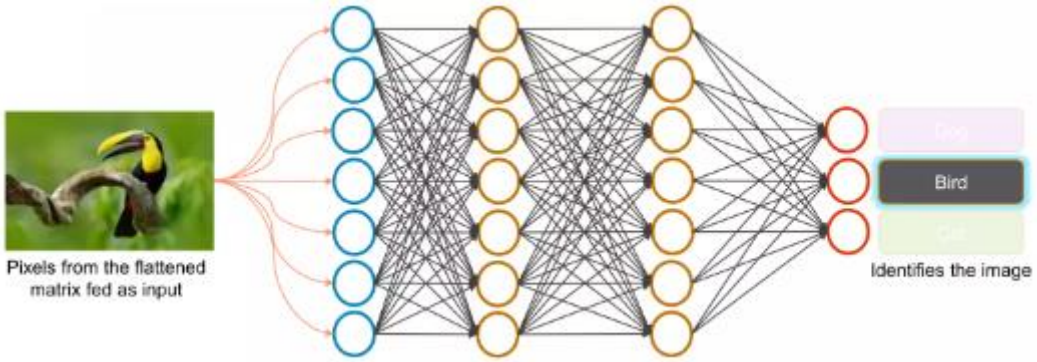


Fig 14: Fully Connected Layer

Entire process:

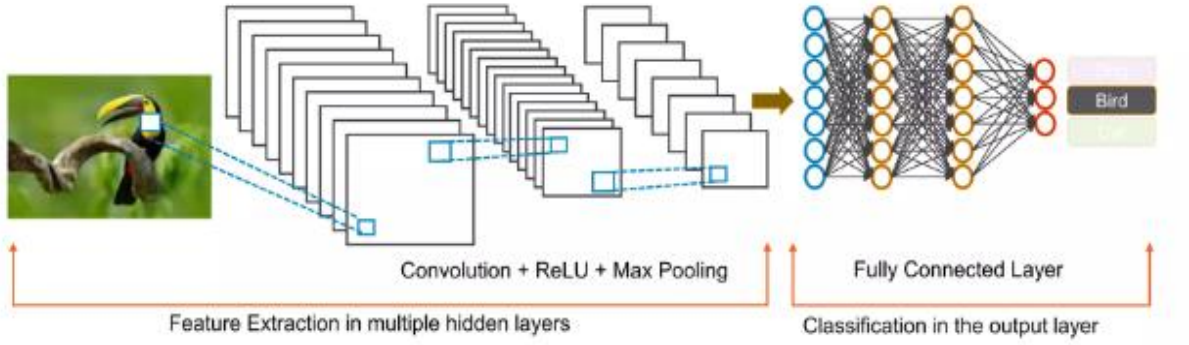


Fig 15: Entire Process in CNN

In summary, neural networks are powerful tools in deep learning, with various types optimized for specific tasks. RNNs, especially LSTM, provide solutions for handling sequential data with long-term dependencies, making them suitable for time series analysis, language processing, and more. Meanwhile, CNNs excel in image recognition tasks through their convolution, ReLU, pooling, and fully connected layers. Together, these neural network architectures contribute significantly to the advancement of artificial intelligence and machine learning technologies.

Details of Database:

For this work, I have made use of NSL-KDD dataset. In it I made use of KDDTest+.txt and KDDTrain+.txt files. NSL-KDD is a new version data set of the KDD'99 data set. It does not include redundant records in the train set, so the classifiers will not be biased towards more frequent records.

- **Training Dataset :**

The KDD+ training dataset consists of 125,973 samples, each containing 116 features. Among these features, 32 are continuous variables, while 84 are binary features representing categorical information in a one-hot encoded format. The dataset is designed to handle multi-class classification tasks with 23 different label categories. These labels represent various types of network intrusions, including malicious activities, and one label denotes "normal" traffic without any intrusion.

To preprocess the data, six continuous features with predominantly zero values were identified and removed from the dataset, as they would not contribute significantly to the classification process. The remaining continuous features underwent scaling, ensuring that they were transformed to a common range between 0 and 1, which aids in effective training of machine learning models. The categorical features in the dataset, specifically the ones indicating protocol, flag, and service information, were one-hot encoded. This transformation converts categorical variables into a binary vector representation, making them suitable for inclusion in machine learning algorithms.

It is important to note that the training dataset does not include any anomalies that are unique to the test dataset. This means that the training data covers a broad range of intrusion scenarios, ensuring the model is exposed to a variety of attack types, enabling it to generalize well when applied to real-world situations.

By following these preprocessing steps, the KDD+ training dataset is well-prepared for training machine learning models to detect and classify network intrusions accurately. The dataset's diversity in label categories and the absence of unique anomalies in the training data contribute to building robust and effective intrusion detection systems for network security applications.

- **Test Dataset :**

The test dataset in the KDD+ environment comprises 22,544 samples, with each sample containing 116 features. Among these features, 32 are continuous variables, while 84 are binary features representing categorical information, transformed using one-hot encoding. The primary objective of this dataset is multi-class classification, as it contains 38 label categories, representing various types of network intrusions, alongside "normal" traffic.

In terms of label distribution, the test dataset consists of 21 labels that are common with the training dataset. These labels reflect network intrusions and normal traffic scenarios encountered during both training and testing phases. Additionally, there are 2 labels exclusively present in the training dataset, meaning these particular intrusion types were not encountered during the test dataset's data collection.

One noteworthy aspect of the test dataset is the presence of 17 labels that are entirely unique to this dataset. These exclusive labels represent anomalies that were not part of the training dataset. This crucial characteristic ensures a realistic evaluation of intrusion detection systems, as it simulates scenarios where new and previously unseen types of network intrusions emerge.

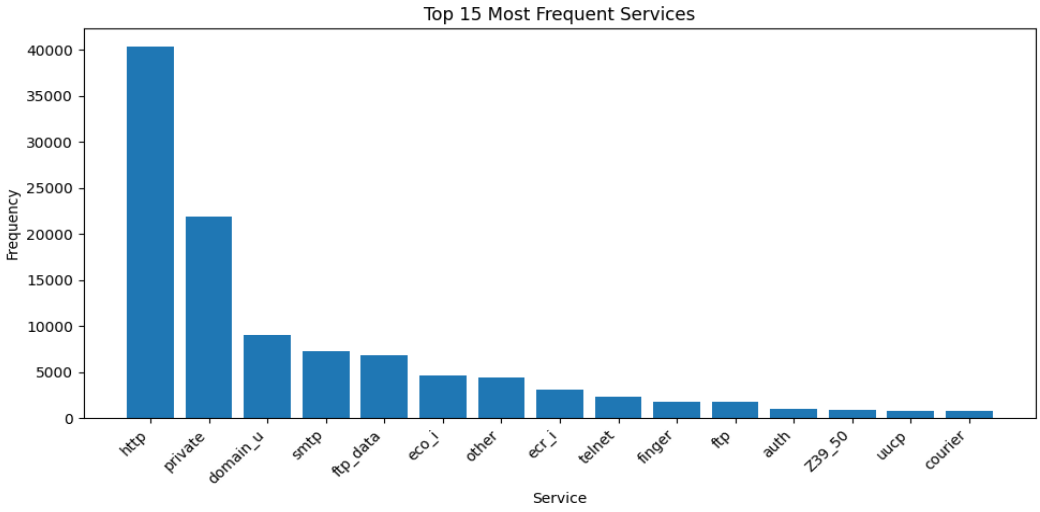


Fig 16: Frequency Distribution of 15 Most Frequent Services

AI/ML model under consideration:

- **Model Description:**

  Different deep learning models have been studied. The model with best detection performance has been a combination of CNN and RNN.

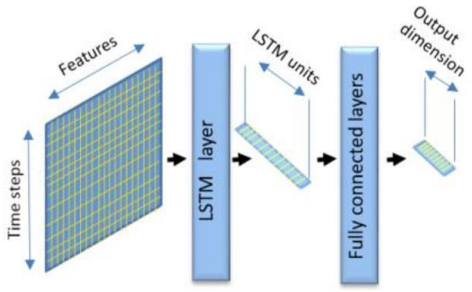


Fig 17: Deep Learning RNN Model

The first model, a simple RNN using LSTM, which is easier to train due to its ability to solve the vanishing gradient problem. The LSTM is trained with a matrix of values containing temporal dimensions and feature vectors. It iterates over these sequential feature vectors along with two additional vectors representing internal hidden and cell states. The output dimension of an LSTM layer is the same as the size of its internal hidden state. Fully connected layers are added at the end of all models.

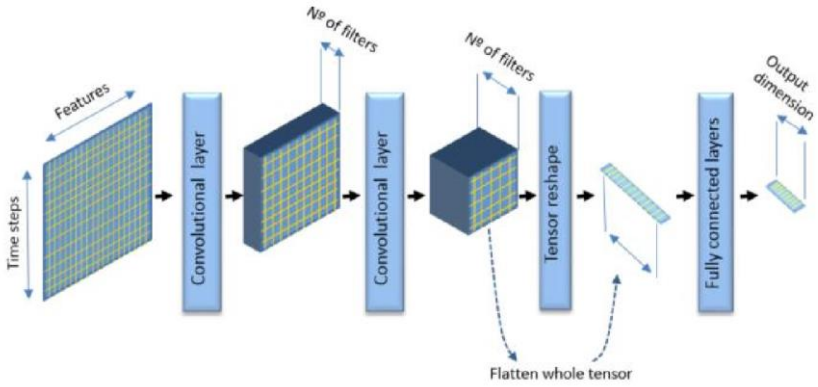


Fig 18: Deep Learning CNN Model

The second model, a pure CNN network initially designed for image processing and classification. The analogy is applied to a different dataset by considering the time-series feature vectors as an image. CNN layers reduce image dimensions while generating a new dimension with a size equal to the number of filters applied. To make the model compatible with the fully connected layers, the tensor is flattened.

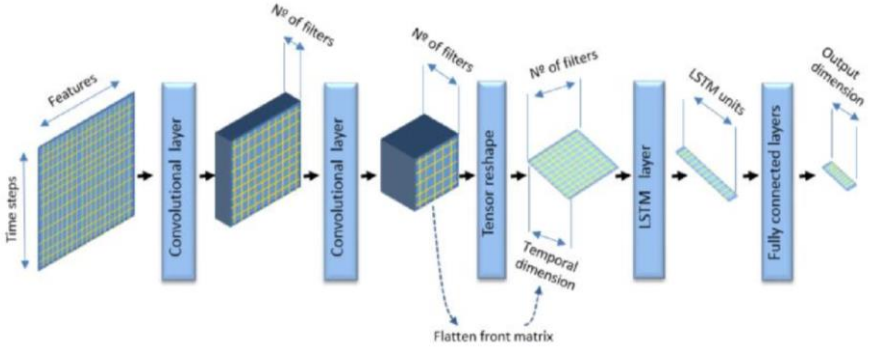


Fig 19: Deep Learning CNN+RNN Model

The third model is a combined model that reshapes the final tensor from chained CNNs into a matrix to act as input to an RNN (LSTM). The last CNN's filter values become equivalent to feature vectors, and the flattened vector serves as the time dimension for the LSTM layer.

Additional layers used in the models are explained:

- Dropout layers[8] provide regularization by randomly setting a percentage of outputs from the previous layer to zero, preventing over-reliance on specific inputs and improving generalization.
- Max pooling layers[9], similar to convolutional layers, use a max filter to down-sample the output, reducing the number of features and computational complexity, thus providing regularization.
- Batch normalization[10] normalizes features at the batch level during training, accelerating convergence and potentially enhancing performance.

These different layers and models are employed to create a comprehensive and effective neural network architecture for the given application.

Implementation and Result:

This section presents the results obtained when applying several deep learning models to NTC. The influence of several important hyper-parameters and design decisions is analyzed, in particular: the model architecture. In order to appreciate the detection quality of the different options, and considering the highly unbalanced distribution of service labels, we provide accuracy for each option.

A. Impact of Network Architecture:

We tested various deep learning architecture models to determine how well they applied to the NTC issue. We have taken into account a 3 models in order to build the various architectures, including RNN only, CNN only, and configuration of a CNN followed by an RNN. We describe the various architectures in Table I, and we show their plotting of training accuracy in Fig. 20. demonstrates that the model CNN+RNN provides the best accuracy result.

The Python packages used in the area are as follows:

1. ***pandas*** (`import pandas as pd`): Pandas is a powerful library for data manipulation and analysis. It is used to read and process data from CSV files and perform various data preprocessing tasks.
2. ***numpy*** (`import numpy as np`): NumPy is a fundamental package for numerical computations in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.
3. ***tensorflow*** (`import tensorflow as tf`): TensorFlow is an open-source deep learning framework developed by Google. It allows users to build and train machine learning models, particularly neural networks, efficiently on GPUs and CPUs.
4. ***Sklearn[33]*** (`from sklearn.model_selection import train_test_split`, `from sklearn.preprocessing import LabelEncoder`): Scikit-learn is a widely used machine learning library in Python. It provides various tools for data preprocessing, feature selection, model evaluation, and more. In the code, it is used for splitting the data into training and testing sets and for label encoding of the target variable.
5. ***tensorflow.keras*** (`from tensorflow.keras.utils import to_categorical`, `from tensorflow.keras.models import Sequential`, `from tensorflow.keras.layers import ...`): TensorFlow's Keras API is a high-level neural networks API that provides an easy-to-use interface for building and training deep learning models. It is built on top of TensorFlow and simplifies the process of defining, compiling, and training neural networks.
6. ***matplotlib*** (`import matplotlib.pyplot as plt`): Matplotlib is a popular data visualization library in Python. It is used in the code to plot the training accuracy of different models over epochs.

Overall, these Python packages are used for data loading, preprocessing, one-hot encoding, model building, training, and plotting in the context of machine learning and deep learning tasks.

The architecture description provided in Table I is as follows: Conv(z,x,y,n,m) stands for a convolutional layer with z filters where x and y are the width and height of the 2D filter window, with a stride of n and SAME

padding if m is equal to S or VALID padding if m is equal to V (VALID implies no padding and SAME implies padding that preserves output dimensions). MaxPool(x,y,n,m) stands for a Max Pooling layer where x and y are the pool sizes, with a stride of n and SAME padding if m is equal to S or VALID padding if m is equal to V (VALID implies no padding and SAME implies padding that preserves output dimensions). BN stands for a batch normalization layer. FC(x) stands for a fully connected layer with x nodes. LSTM(x) stands for an LSTM layer where x is the dimensionality of the output space; in the case of several LSTM in sequence, each LSTM, except the last one, will return the successive recurrent values which will be the entry values to the following LSTM. DR(x) stands for a dropout layer with a dropout coefficient equal to x.

In all cases, the training was done with a 10 number of epochs. Considering an epoch as a single pass of the complete training dataset through the training process. All the activation functions were Rectified Linear Units (ReLU) with the exception of the last layer with Softmax activation.

Model	Architecture Details
RNN	LSTM(100)-FC(100,'relu')-FC(100,'relu')-Flatten()-FC(len(label_mapping), 'softmax')
CNN	Conv1D(32, 4, 1, 'same', 'relu')-MaxPool(3, 1, 'same')- Flatten()-FC(200, 'relu')-FC(len(label_mapping), 'softmax')
CNN+RNN	Conv1D(32, 3, 'relu', input_shape=(X_train.shape[1], 1))--MaxPool(2)-LSTM(32, True)-FC(64, 'relu')-Flatten()-FC(len(label_mapping), 'softmax')

Table I. Details of deep learning network models applied to NTC problem

Steps taken in implementation of code:

1. Load the training and test data from CSV files.
2. Define the label_mapping dictionary to map class names to numerical labels.
3. Preprocess the training and test data, including label encoding and one-hot encoding of categorical variables.
4. Map the test data labels to the same set of labels as the training data and remove rows with missing labels from the test data.
5. Split the data into features (X) and labels (y) for both training and test datasets.
6. Convert the labels to one-hot encoded format.
7. Reshape the input data to be compatible with the CNN and RNN models (1D data).
8. Define the CNN model
9. Compile the CNN model
10. Define the RNN model
11. Compile the RNN model
12. Define the CNN+RNN model
13. Compile the CNN+RNN model
14. Train all three models (CNN, RNN, CNN+RNN) with the training data and store the training history for accuracy.
15. Extract the accuracy from the training history of each model.
16. Plot the accuracy of the three models over the training epochs.

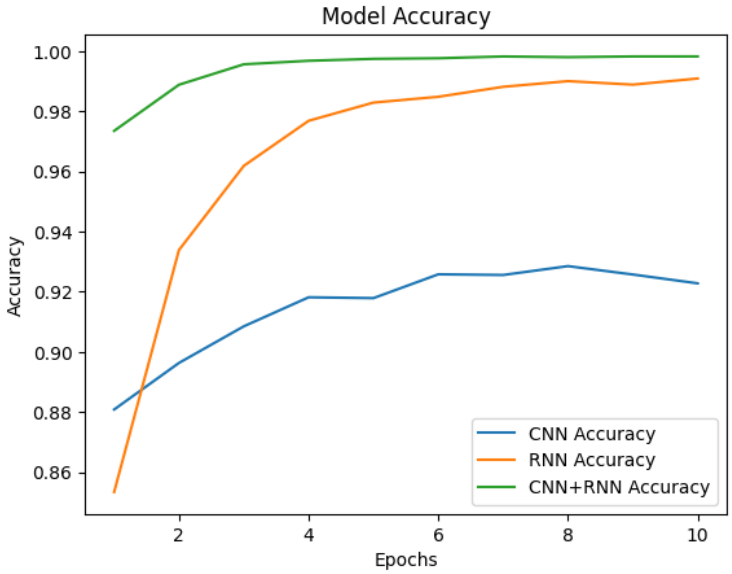


Fig 20. plotting of training accuracy

For the CNN Model: The training accuracy starts from around 88.01% and steadily increases with each epoch, reaching around 91.32% by the 10th epoch.
For the RNN Model: The training accuracy starts from around 84.32% and increases with each epoch, reaching around 98.40% by the 10th epoch.
For the CNN+RNN Model: The training accuracy starts from around 96.96% and steadily increases with each epoch, reaching around 99.75% by the 10th epoch.

Overall, observation is that, all three models shows an improvement in training accuracy and a decrease in training loss during the training process. The CNN+RNN model achieves the highest training accuracy, followed by the CNN and RNN model.

B. Impact of Feature

In this section we will see the influence of the features employed in the learning process. Table II shows the importance of features by analyzing the detection metrics as we remove different features. The first column in Table II gives the features that are used to train the model (grouped in feature sets) and the right columns the usual aggregate performance metrics for the detection process. The chart presents the same results in a different format, to make it easier to compare different feature sets. Model CNN+RNN was used to obtain the result presented in Table 2,but the same relationship between results and feature was maintained when repeating this same study with the other model.

Set	Feature Names	Accuracy	Precision	Recall	F1-score
1	duration, src_bytes, dst_bytes, protocol_type	0.562175	0.461320	0.562175	0.458769
2	duration, src_bytes, dst_bytes, service	0.633022	0.586339	0.633022	0.594340
3	duration, src_bytes, dst_bytes, flag	0.541298	0.461897	0.541298	0.452624
4	duration, src_bytes, dst_bytes, land	0.527644	0.423962	0.527644	0.402675
5	duration, src_bytes, dst_bytes, logged_in	0.529867	0.355983	0.529867	0.408087

Table 2: Classification Performance metrics vs. Features (model CNN+RNN)

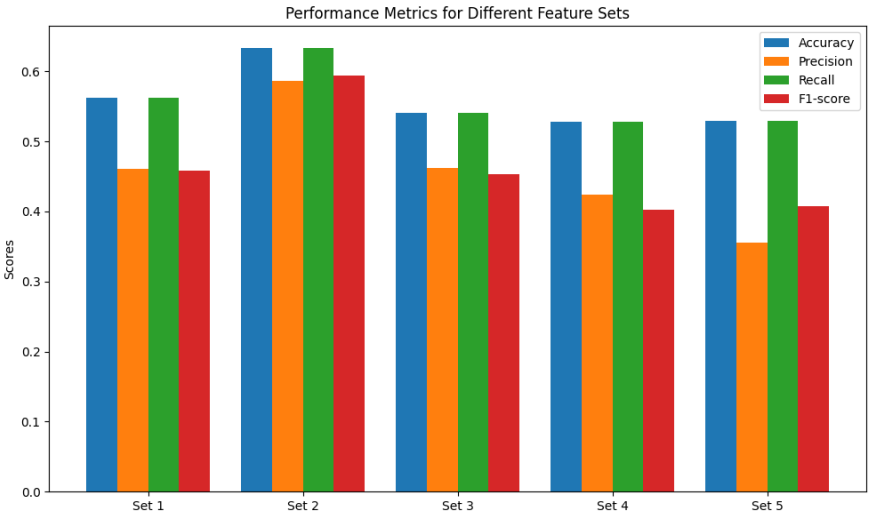


Fig 21: Classification Performance Metrics vs. Features (model CNN+RNN)

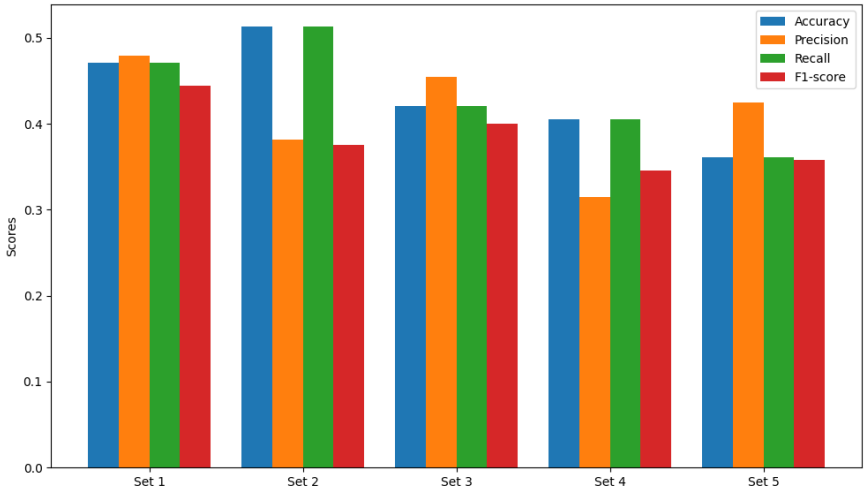


Fig 22: Classification Performance Metrics vs. Features (model CNN)

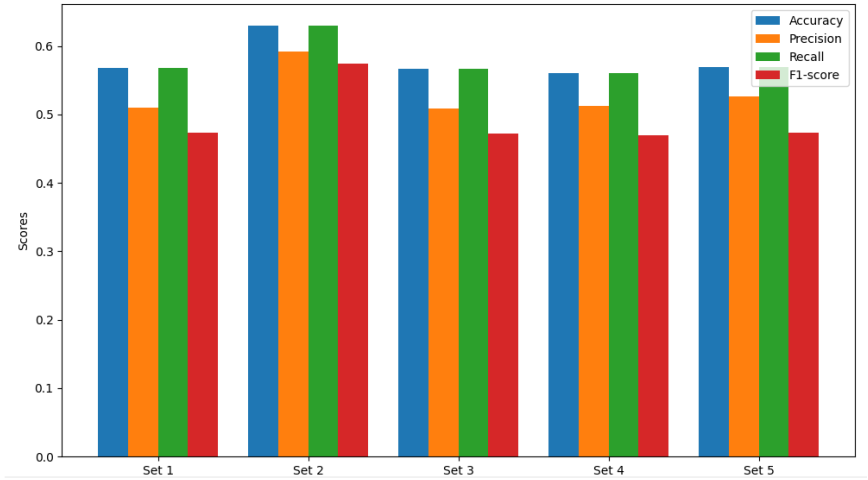


Fig 23: Classification Performance Metrics vs. Features (model RNN)

List of the Python packages used in the code:

1. ***pandas:*** Used for data manipulation, reading and writing data in DataFrame format.
2. ***numpy:*** Used for numerical operations and array manipulation.
3. ***sklearn:*** Scikit-learn library for machine learning tasks, including data preprocessing, train-test split, and evaluation metrics.
4. ***keras:*** Keras is a high-level neural networks API, used for building and training deep learning models. Note that Keras is now integrated into TensorFlow as tf.keras.
5. ***matplotlib:*** Used for data visualization, specifically for creating the bar plot to display performance metrics.

Sure! Here's a step-by-step explanation of the code:

1. Import necessary libraries: The code starts by importing the required libraries, including pandas, numpy, scikit-learn, Keras, and matplotlib.

2. Load the dataset: The code loads the dataset from the CSV file "train_data.csv" into a Pandas DataFrame named "data".

3. Extract common features: Three common features ("duration", "src_bytes", "dst_bytes") are extracted from the dataset into a DataFrame named "common_features".

4. Prepare labels: The target labels are extracted from the last column of the DataFrame "data" and are one-hot encoded using the "to_categorical" function. A label mapping is also created to map labels to indices.

5. Define the CNN+RNN model: The function "rnn_cnn_model" defines the architecture of the CNN+RNN model using the Keras Sequential API.

6. One-hot encode categorical features: The categorical features ("protocol_type", "service", "flag", "land", "logged_in") are one-hot encoded separately using OneHotEncoder from scikit-learn.

7. Concatenate numerical and encoded categorical features: The numerical features are concatenated with the encoded categorical features to create five different feature sets (X_set1, X_set2, X_set3, X_set4, X_set5).

8. Split the data: Each feature set and the one-hot encoded labels are split into training and testing sets using "train_test_split" from scikit-learn.

9. Train and evaluate the models: The function "train_and_evaluate" is defined to train and evaluate the CNN+RNN model on each feature set. It returns accuracy, precision, recall, and F1-score metrics for evaluation.

10. Store results in a DataFrame: The performance metrics for each feature set along with their feature names are stored in a DataFrame named "results_df".

11. Display the results: The "results_df" DataFrame is printed to display the performance metrics for each feature set.

12. Plot the results: The performance metrics (accuracy, precision, recall, F1-score) for each feature set are plotted using Matplotlib to visualize and compare the model's performance.

Overall, the code performs multi-class classification using the CNN+RNN model on different feature sets and evaluates the model's performance on each set using various metrics. The results are stored in a DataFrame and plotted for visualization.

Conclusion:

In conclusion, this research paper presents a novel Network Traffic Classification (NTC) technique specifically designed for the Internet of Things (IoT) traffic analysis. By combining Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), the proposed approach achieves improved detection results without the need for complex feature engineering.

The key contributions and findings of this research can be summarized as follows:

- CNNs are excellent for capturing spatial patterns and features in data, such as images or time series. In the context of network traffic classification, CNNs can effectively learn local patterns from packet payloads, header information, or other relevant features present in the data. They excel at feature

extraction through the use of convolutional layers and pooling, making them suitable for tasks where local patterns are important.

- CNNs are particularly useful when dealing with data that exhibit spatial or local relationships. In the context of IoT network traffic, where packets might contain important patterns that are relevant to the classification task, CNNs can be employed to identify and learn these patterns effectively.
- RNNs are well-suited for tasks involving sequential data and time dependencies. In network traffic classification, RNNs can take into account the sequential nature of packet transmissions and capture temporal patterns. They have memory that allows them to maintain information from previous time steps, making them capable of handling dynamic and time-sensitive data.
- RNNs are beneficial when the temporal order of the data plays a crucial role in the classification task. In IoT network traffic, where the sequence of packet arrivals and interactions matters, RNNs can effectively learn and model these dependencies.
- Combining CNN and RNN models can leverage the strengths of both architectures. CNNs can capture local patterns and features from the data, while RNNs can handle sequential dependencies. By using a hybrid model, the classifier can potentially benefit from both spatial and temporal information present in the network traffic data.
- The research explores the combination of CNNs and RNNs to create a powerful and effective neural network architecture for NTC. CNNs excel in feature extraction from IoT traffic data, while RNNs capture sequential dependencies for time-series analysis.
- The CNN model successfully applies image-processing concepts to vector time-series data, extending the capabilities of CNNs beyond text and audio processing. The combined CNN and RNN model yields the best detection results, outperforming alternative techniques reported in previous works.
- The CNN+RNN hybrid is often preferred over using RNN followed by CNN because CNNs are inherently better at feature extraction from spatial data (which is essential in network traffic), while RNNs are better at handling sequential dependencies. When using CNN followed by RNN, the RNN might not receive the most informative features extracted by the CNN, potentially limiting its performance. In contrast, the hybrid architecture can learn relevant spatial features using CNN and then model temporal dependencies effectively using RNN, leading to improved classification accuracy.

Overall, the results demonstrate the effectiveness of the CNN+RNN integration for IoT Network Traffic Classification, paving the way for improved management and monitoring of IoT networks. The research contributes to a smoother and more efficient IoT experience by segregating traffic and understanding the behavior of different IoT devices and services. Future work may involve further optimization of the models, testing on larger datasets, and real-world deployment to validate the technique's performance in IoT environments.

References:

[1] B. Ng, M. Hayes and W. K. G. Seah, "Developing a traffic classification platform for enterprise networks with SDN: Experiences & lessons learned," 2015 IFIP Networking Conference (IFIP Networking), Toulouse, 2015, pp. 1-9.

[2] A. Sivanathan, D. Sherrat, H. Habibi Gharakheili, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman , “Characterizing and Classifying IoT Traffic in Smart Cities and Campuses”, IEEE INFOCOM Workshop on SmartCity: Smart Cities and Urban Computing, USA, May 2017.

[3] T. Nguyen and G. Armitage, “A survey of techniques for internet traffic classification using machine learning,” IEEE Commun. Surv. Tutorials, vol. 10, no. 4, pp. 56–76, 2008.

[4] J. Zhang, X. Chen, Y. Xiang, W. Zhou, and J. Wu, “Robust Network Traffic Classification,” IEEE/ACM Trans. Netw., vol. 23, no. 4, pp. 1257–1270, 2015.

[5] Behnke, Sven, “Hierarchical Neural Networks for Image Interpretation”, Volume 2766 of Lecture Notes in Computer Science, Springer-Verlag, 2003

[6] Zachary C. Lipton, John Berkowitz, Charles Elkan (2015), “A Critical Review of Recurrent Neural Networks for Sequence Learning”, arXiv:1506.00019 [cs.LG]

[7] Klaus Greff; Rupesh Kumar Srivastava; Jan Koutník; Bas R. Steunebrink; Jürgen Schmidhuber (2015). "LSTM: A Search Space Odyssey". arXiv:1503.04069

[8] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. “Dropout: a simple way to prevent neural networks from overfitting”. J. Mach. Learn. Res. 15, 1 (January 2014), 1929-1958.

[9] Chen-Yu Lee, Patrick W. Gallagher, Zhuowen Tu (2015), “Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree”, arXiv:1509.08985 [stat.ML]

[10] Sergey Ioffe, Christian Szegedy (2015), “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, arXiv:1502.03167 [cs.LG]

[11] T. Auld, A. W. Moore, and S. F. Gull, “Bayesian Neural Networks for Internet Traffic Classification,” IEEE Trans. Neural Networks, vol. 18, no. 1, pp. 223–239, Jan. 2007.

[12] X. Xie, B. Yang, Y. Chen, L. Wang and Z. Chen, "Network Traffic Classification Based on Error-Correcting Output Codes and NN Ensemble," 2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery, Tianjin, 2009, pp. 475-479.

[13] Chen, Z., Wang, H., Abraham, A., Grosan, C., Yang, B., Chen, Y., Wang, L., “Improving Neural Network Classification Using Further Division of Recognition Space”. International Journal of Innovative, Computing, Information and Control 5(2) (2009)

[14] Herrero, A., Corchado, E., Gastaldo, P., Zunino, R., “Neural projection techniques for the visual inspection of network traffic”. Neurocomputing, Volume 72, Issue 16, Pages 3649-3658, 2009.

[15] Kothari, A., Keskar, A., “Rough Set Approaches to Unsupervised Neural Network Based Pattern Classifier”, Advances in Machine Learning and Data Analysis, Springer Netherlands, Dordrecht, pp. 151163, 2010.

[16] W. Zhou, L. Dong, L. Bic, M. Zhou and L. Chen, "Internet traffic classification using feed-forward neural network," 2011 International Conference on Computational Problem-Solving (ICCP), Chengdu, 2011, pp. 641-646.

[17] A Moore D Zuev L. Crogan "Discriminators for use in flow-based classification", Technical Report RR-05-13. Department of Computer Science Queen's Mary University, 2005.

[18] Bereket Mathewos, Marco Carvalho, and Fredric Ham. 2011. “Network traffic classification using a parallel neural network classifier architecture”. In Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research (CSIIRW '11), Frederick T. Sheldon, Robert Abercrombie, and Axel Krings (Eds.). ACM, New York, NY, USA, , Article 33 , 1 pages.

[19] Kim, H., Claffy, K., Fomenkov, M., Barman, D., Faloutsos, M., Lee, K.: “Internet traffic classification demystified: myths, caveats, and the best practices”, In Proceedings of the 2008 ACM CoNEXT Conference (CoNEXT '08). ACM, New York, NY, USA, pp. 11:1–11:12, 2008.

[20] M. Shafiq, Xiangzhan Yu, A. A. Laghari, Lu Yao, N. K. Karn and F. Abdessamia, "Network Traffic Classification techniques and comparative analysis using Machine Learning algorithms," 2016 2nd IEEE International Conference on Computer and Communications (ICCC), Chengdu, China, 2016, pp. 2451-2455.

[21] C. Wang, T. Xu and X. Qin, "Network Traffic Classification with Improved Random Forest," 2015 11th International Conference on Computational Intelligence and Security (CIS), Shenzhen, 2015, pp. 78-81. doi: 10.1109/CIS.2015.27

[22] Jun Zhang, Chao Chen, Yang Xiang, Wanlei Zhou, and Athanasios V. Vasilakos, “An Effective Network Traffic Classification Method with Unknown Flow Detection”, IEEE Transaction on Network and ServiceManagement, Vol 12, Dec 2013

[23] Y.-s. Lim, H.-c. Kim, J. Jeong, C.-k. Kim, T. T. Kwon, and Y. Choi, “Internet traffic classification demystified: on the sources of the discriminative power,” In Proceedings of the 6th International COnference (Co-NEXT '10). ACM, New York, NY, USA, pp. 9:1– 9:12. 2010.

[24] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson, “Offline/realtime traffic classification using semi-supervised learning,” Performance Evaluation, vol. 64, no. 9-12, pp. 1194–1213, Oct. 2007.

[25] N. Williams, S. Zander, and G. Armitage, “A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification,” SIGCOMM Comput. Commun. Rev., vol. 36, pp. 5–16, Oct. 2006.

[26] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield, “Class-of-service mapping for QoS: a statistical signature-based approach to IP traffic classification,” in Proc. 2004 ACM SIGCOMM Conference on Internet Measurement, pp. 135–148.

[27] A. W. Moore and D. Zuev, “Internet traffic classification using Bayesian analysis techniques,” SIGMETRICS Perform. Eval. Rev., vol. 33, pp. 50–60, June 2005.

[28] S. Hao, J. Hu, S. Liu, T. Song, J. Guo and S. Liu, "Network traffic classification based on improved DAG-SVM," 2015 International Conference on Communications, Management and Telecommunications (ComManTel), DaNang, 2015, pp.256-26

[29] B. Yamansavascilar, M. A. Guvensan, A. G. Yavuz and M. E. Karsligil, "Application identification via network traffic classification," 2017 International Conference on Computing, Networking and Communications (ICNC), Santa Clara, CA, 2017, pp. 843-848.

[30] Z. Yuan and C. Wang, "An improved network traffic classification algorithm based on Hadoop decision tree," 2016 IEEE International Conference of Online Analysis and Computing Science (ICOACS), Chongqing, 2016, pp. 53-56. doi: 10.1109/ICOACS.2016.7563047

[31] L. Deri, M. Martinelli, T. Bujlow, and A. Cardigliano, “nDPI: Opensource high-speed deep packet inspection,” in 2014 International Wireless Communications and Mobile Computing Conference (IWCMC), 2014, pp. 617–622.

[32] T. Bujlow, V. Carela-Español, and P. Barlet-Ros, “Independent comparison of popular DPI tools for traffic classification,” Comput. Networks, vol. 76, pp. 75–89, 2015.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, “Scikit-learn: Machine Learning in Python”, Journal of Machine Learning Research, vol. 12, pp. 2825-2830, 2011.

[34] T. N. Sainath, O. Vinyals, A. Senior and H. Sak, "Convolutional, Long Short-Term Memory, fully connected Deep Neural Networks," 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), South Brisbane, QLD, 2015, pp. 4580-4584.

[35] Y. Xiao, K. Cho, “Efficient Character-level Document Classification by Combining Convolution and Recurrent Layers”, arXiv:1602.00367v1 [cs.CL] 1 Feb 2016

[36] Y. Meidan, M. Bohadana, A. Shabtai, J. D. Guarnizo, M. Ochoa, N. O. Tippenhauer, and Y. Elovici. 2017. “ProfilIoT: a machine learning approach for IoT device identification based on network traffic analysis”. In Proceedings of the Symposium on Applied Computing (SAC '17). ACM, New York, NY, USA, 506-509

[37] S. Althunibat, A. Antonopoulos, E. Kartsakli, F. Granelli and C. Verikoukis, "Countering Intelligent-Dependent Malicious Nodes in Target Detection Wireless Sensor Networks," in IEEE Sensors Journal, vol. 16, no. 23, pp. 8627-8639, Dec.1, 2016.

[38] M. Grajzer, M. Koziuk, P. Szczechowiak and A. Pescape, "A MultiClassification Approach for the Detection and Identification of eHealth Applications," 2012 21st International Conference on Computer Communications and Networks (ICCCN), Munich, 2012, pp. 1-6.