Network Trafffic Scenario prediction Challenge by ITU

Author: Adeyinka Michael Sotunde (aka MICADEE)

Email:shotundeadeyinka@yahoo.com

1.0 Introduction:

Traffic scenario classification can be used in multiple service scenarios to accurately optimize network parameters or take different management and control measures based on the specific scenario. For example, in intelligent operation and maintenance, different fault types can be identified based on the traffic performance caused by each fault. In intelligent congestion control, rate control parameters can be configured based on traffic performance to ensure high throughput, low delay, and no packet loss. However, the status parameters of traffic in different scenarios may not differ significantly due to limitations in the collection device. Moreover, the number of scenarios within a given duration is uncertain, and random switching between scenarios can make it difficult to identify them. In some real-time decision situations, a response time of milliseconds or even microseconds may be required, which presents a greater challenge in solving the problem.

During transmission and forwarding processes, network traffic may be sent to the same port queue by multiple ports. As the output rate of the forwarding device is limited by the port bottleneck bandwidth, the input rate of traffic is higher than the output rate, resulting in the need for the forwarding device to allocate cache space to store burst traffic. This creates a cache queue at the egress port of the forwarding device. The queue length in the cache queue changes based on different types of forwarding traffic. For example, queue flaps caused by different stream ratios and different loads are different. All these factors are different in various service types such as web search and distributed computing. As service layer applications change, the traffic characteristics received by the same queue also change.

2.0 Dataset Overview:

There are some key status parameters in form of datasets used in this analysis representing the key components or factors in this traffic scenario classification project, such as input rate (v_in), output rate (v_out), real-time length of the cache queue (q), and corresponding time (t), are collected at fixed time intervals. The output rate (v_out) is limited by the port bottleneck rate (v_max) where v_out <= v_max, while the input rate is affected by the burst and regulation protocol. So the maximum of input rate may be higher than the bottleneck rate. Here, the input and output rates are measured by the amount of traffic received and sent during each fixed time interval.

2.1 Objective of the Analysis

The primary objective of this traffic scenario classification task is to build a model based on the training set data to predict the traffic scenario for unknown traffic at each moment. For instance, in this context, an "unknown scenario" refers to a scenario with known traffic but unclear which of the known scenarios it belongs to, rather than a completely new scenario that has never occurred before.

3.0 Methodology

3.1 Data Preprocessing: The input data consists of CSV files with various columns excluding the target column. Each column represents an input channel effectively creating a 1D signal with three (or four if time is included) features. These features were engineered by applying (np.log1p) to each. And more so, since each signal varies in length, it is crucial to handle these variances to avoid errors during data loading. A batch size of 1 per CSV file is recommended to mitigate this issue.

3.2 Model Architecture: A 1D CNN UNET architecture is the primary model employed in this phase with an open-source pytorch lightning framework. This architecture demonstrated fast training and the potential for achieving high scores. The model takes advantage of the multiple input channels treating the 78 examples as 78 different cases akin to working with images.

3.3 Model Optimization: several strategies were applied to enhance model performance:

- Increase the Number of Stages and Layers: Standard UNET models have four stages; however, increasing the number of stages can capture more complex patterns.
- Increase Stride: A larger stride allows the model to capture distant relationships between points, which is valuable for understanding network traffic patterns.

- Increase Kernel Size: Expanding the kernel size increases the window of application for each convolutional function enabling better feature extraction.

3.4 Hyperparameters Overview:

- **max_epochs:** Set to 15, this hyperparameter controls the maximum number of training epochs. It determines how many times the entire dataset is passed forward and backward through the neural network. A higher value allows the model to learn more but may risk overfitting if not controlled.
- **patience:** With a value of 10, this hyperparameter determines the number of epochs to wait for improvement in the validation loss before early stopping. Early stopping helps prevent overfitting by halting training when the model's performance on the validation set starts degrading.
- **n_splits:** A value of 10 for this hyperparameter indicates the number of splits in k-fold cross-validation. Cross-validation helps assess the model's performance on multiple subsets of the dataset, ensuring robustness and reducing the risk of overfitting.
- **layer_n:** Set to 32, this hyperparameter determines the number of layers used in the neural network architecture. It plays a crucial role in defining the network's depth and complexity.
- **lr (Learning Rate):** A learning rate of 1e-3 was chosen. The learning rate controls the step size during gradient descent optimization. Finding an appropriate learning rate is essential for model convergence.

3.5 Convolutional Neural Network Architecture:

After carefully implemented a very strong Convolutional Neural Network Architecture, thus to optimize this model architecture, several key aspects that is important was focused on in this model:

Pooling Stride (pool_stride): The pool stride hyperparameter was set to 5. Pooling layers down sample the input data, reducing its spatial dimensions. The stride defines the step size at which the pooling operation is applied. A larger stride can quickly capture information but may lose fine-grained details. In this case, a stride of 5 was selected to balance capturing essential features while retaining sufficient detail.

Kernel Size (kernel_size): The kernel_size hyperparameter was set to 7. Kernel size determines the receptive field of each convolutional layer. A larger kernel size allows the model to capture more extensive patterns in the input data. In this architecture, a kernel size of 7 was chosen to provide a moderate receptive field, suitable for extracting meaningful features.

Network Depth (depth) and Block Depth (block_depth): The network depth and block depth were experimented to capture complex patterns effectively. A depth of 3 indicates that I explored

networks with three stages, while block_depth of 2 suggests that each stage consisted of two layers. Increasing the network depth allows the model to learn hierarchical features, while block depth controls the complexity of each stage.

Dropout (dropout): A dropout rate of 0.08 was applied which also happen to be the best dropout to mitigate overfitting. Dropout is a regularization technique that randomly drops a fraction of neurons during training. This helps prevent the model from relying too heavily on specific neurons and encourages more robust learning.

4.0 Experimental Results

The journey to achieving competitive scores involved iterative improvements and adaptations. Key insights derived from the submissions are shared:

Score of 0.76352: In my initial attempt, Transformer-like Attention layers were incorporated into the highest stage of aggregation within the UNet to capture interactions between distant time intervals. However, this approach did not yield the desired results leading it to pivot towards a pure CNN UNet.

Score of 0.77365: After removing the attention component, I focused on optimizing hyperparameters through cross-validation. This effort significantly improved my model score; however, it also highlighted the substantial computing power required for extensive cross-validation runs.

Score of 0.775143: Further improvements came from the integration of hand-engineered features, calculated using a sliding window of fixed time interval length. These features enhanced the model's performance, demonstrating the value of feature engineering.

Score of 0.777838: An insightful observation during model output analysis led to our highest score. It was noticed that predictions were more accurate when the target was near the center of the input subsequences. To address this, it was experimented with using a batch size of 1 and inputting the full time series as a whole. Additionally, I implemented a prediction aggregation approach to mitigate performance drop-offs at the edges of the prediction window. This innovative solution significantly improved the overall model performance.

The rigorous hyperparameter tuning and optimization efforts led to notable improvements in model performance. Finally, a score of **0.777838** on Public Leaderboard (**0.777310** on Private Leaderboard) was achieved by carefully selecting and fine-tuning these hyperparameters. This score demonstrated the effectiveness of the approach in extracting valuable insights from the network traffic signals. Furthermore, other configurations such as a max_epochs of 15 with a score of 0.76352, max_epochs of 15 with a score of 0.775143997, and max_epochs of 13 with a score of 0.7736 were explored indicating the robustness of the methodology.

Conclusion:

The Network Traffic Scenario Prediction Challenge presented a formidable signal segmentation task, closely resembling the segmentation of ECG signals. This technical report has detailed the strategies, methodologies, and insights acquired in the course of this journey to tackle this challenge effectively. The overarching conclusion underscores the significance of creative problem-solving and meticulous model optimization when working with large, time-based datasets.