REPORT ON THE ITU NETWORK TRAFFIC SCENARIO PREDICTION CHALLENGE

Martin Wosnitzka

Abstract *– This is the report corresponding to the highest-scoring submission in the Network Traffic Scenario Prediction Challenge by ITU. It uses a one-dimensional UNET to extract features at a high level. Some hand-engineered time series features were added to the input that were suspected to improve performance. Additionally, a simple postprocessing routine was employed that summarizes the differing model predictions with regard to the target's relative position within a particular subsequence input. The final submission was made by a model ensemble and underwent a nearest neighbor smoothing routine.*

1. INTRODUCTION AND DATASET

The Network Traffic Scenario Prediction Challenge is a competition created by ITU. Its goal is to classify network traffic scenarios for each moment in time based only on the data input and output rates as well as cache queue size of a specific device. More accurate scenario classification can help to optimize network parameters and management policies, improving, among others, the handling of faults and congestions of the network. More details on the problem statement and data as well as the rules and leaderboard of the competition can be found on Zindi, the host of this competition.

The dataset used for training consists of 78 long time series of three channels corresponding to the input rate, output rate and cache queue size. The target is a time series itself indicating the current scenario for each time step. Each of the 78 time series includes many scenarios as the time series themselves are around 100,000 to 150,000 time steps long while the average scenario length is approximately 1250. There are 12 different scenarios labeled 0 through 11 that can occur. Figure 1 shows a short subsequence of one of the time series. The goal is to correctly predict the target scenario time series. The evaluation metric is accuracy.

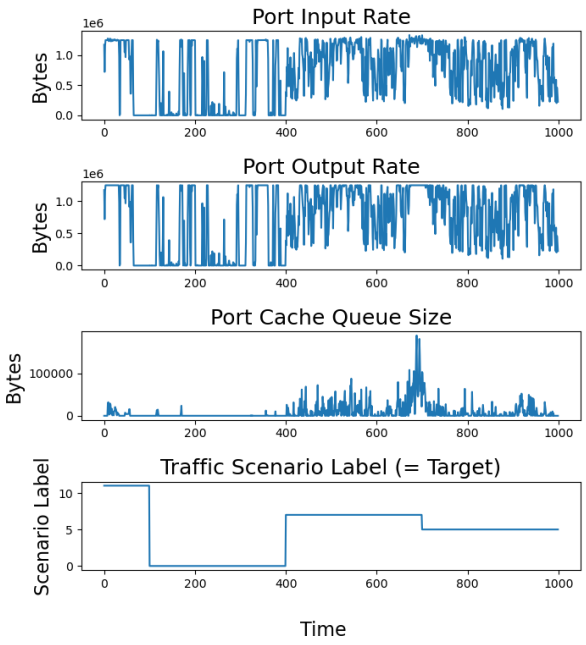

Fig. 1 – Example subsequence of a time series including all input channels and the target scenario label.

2. MODEL

From the fast fluctuation of the input features it is appearant, that it is essential to take the input context before and after each individual target point into account to have a chance at predicting the current scenario. At the same time, the target scenarios are rather long relative to the measuring frequency (the shortest few being 300 time steps long). Thus, a fast local contraction and feature calculation of the input sequences is needed, while still being able to predict a target for each individual point. The UNET-structure fits these requirements very well, as the network is able to quickly pool the sequence, and calculate high-level features on the encoder side, in order to subsequently upsample these features and create predictions on the original scale on the decoder side. Originally developed for (2D-)image segmentation, one-dimensional versions of UNET have successfully been applied to time series before.

Figure 2 shows the detailed architecture of the model applied during the competition. After testing several parameter configurations the following key choices of parameters were made.

Three blocks of encoder-decoder pairs have been used. The encoder blocks consisting of two CNN-layers each, while the decoder blocks only contain one CNN-layer. The number of channels has been chosen to be 16 on the lower levels and 32 on the highest level. While the number of blocks, layers and channels is small, a comparatively large kernel size of 11 has been chosen for all convolutional layers. This trade-off accounts for the dataset's size on

the one hand and the higher quality of features calculated across a rather large window of values of the time series on the other hand. When choosing the pooling and upsampling strides the need to include features of high distance contrasts the loss of information when pooling too many values at once. A value of 5 for all strides proved to be a good balance. In order to further improve the visibility of distant features, the high-level transition part between encoder and decoder is the deepest block of all consisting of three layers. Wherever needed, zero-padding is used to keep the sequence length constant. Note that an earlier version of the model used a simple linear interpolation layer instead of the transposed convolution layer for the upsampling of features. It was later changed in an attempt to increase reproducibility when using deterministic algorithms in pytorch. There was no notable effect on performance with this change.

The model takes subsequences of length 2500 as inputs. This choice will be a topic of discussion later. The data input and output rates are divided by a factor of 10^6 and the cache queue size is log-transformed in order to make the input value ranges more manageable. Some feature engineering has been done by hand resulting in a very strong feature that is added to the input. Together with the three original features these four channels are called the direct input. Beyond the direct input the model takes a separate input of six input channels. All six are features found during feature engineering that exhibited positive effects on performance on an auxiliary task, albeit less significant than the new direct input feature. Using a convolution of kernel size 1 the six features are embedded into a two dimensional feature space before concatenating them with the direct input. This results in an input of six channels that are fed into the UNET. The hand engineered features are presented and discussed in the following section.

The model was trained for 180 epochs on all 78 availabe time series that were randomly split up into subsequences of length 2500 every epoch. After each epoch the performance on the training set was evaluated and the best performing iteration was chosen. A learning rate of 10^{-4} was used with an AdamW optimizer of weight decay 0.01. For the final ensemble three copies of the model were trained with slightly differing parameters. These three slight deviations from the model presented here are explained in the corresponding section below.

3. FEATURE ENGINEERING

The original data contains only three features, two of which - the data input and output rates - are very closely related (Pearson correlation coefficient of > 0.99). In order to enrich the input to the model some hand engineered time series features were tested. For the sake of training speed and to concentrate on the model's ability to correctly classify a given scenario, a separate classifier model was created: The encoder part of the model shown in figure 2 was extracted and a global maximum pooling layer was added after the third block. Then, subsequences of length 300 (= minimum scenario length) corresponding to only one fixed scenario were fed into the model and the model was trained to solve the multi-label classification task of identifying the right scenario given the knowledge that the whole subsequence was part of one single scenario. A 4-fold cross validation was done, training the model for 50 epochs each several times, testing different features on whether they would improve the models accuracy or not.

Out of all the features tested (around 15 in total), there were six features that showed to improve accuracy by a bit and one feature that showed to improve accuracy by a significantly wider margin than any of the other features tested. The later one was added to the direct input to the UNET model while the other six features were bundled up to form the separate input to the UNET model. The following subsections explain these seven features.

3.1 Separate input features

Out of the seven features picked, five have in common that they were calculated using a time window of the series' 25 most recent values and calculating different statistics on this window. That is, given a time series $(X_i)_{1\leq i\leq n}$, the new feature Y_k corresponding to the point in time k $(1 \leq k \leq n)$ is calculated as a function of the 25 input values $X_{k-24}, .., X_k$. In order to be able to do this with all indices, we set $X_i = 0$ if $i < 1$ for all input series.

The last two features take all values $(X_i)_{i\leq k}$ into account. Note that the particular value of 25 for the time window is a heuristic choice and has not been optimized. One might be able to improve performance slightly by tuning this particular parameter.

The following six features have been chosen to make up the separate input to the UNET model:

- **Standard deviation of input rate:** The standard deviation calculated across the 25 most recent values of the input rate.

$$Y_k^{std\_portIn} = \mathit{Std}\left(\{X_i^{portIn}\}_{(k-24)\leq i\leq k}\right)$$

  where *Std* is the usual standard deviation operator.

- **Standard deviation of Cache Queue Size:** The standard deviation calculated across the 25 most recent values of the cache queue size.

$$Y_k^{std\_qSize} = \mathit{Std}\left(\{X_i^{qSize}\}_{(k-24)\leq i\leq k}\right)$$

  where *Std* is the usual standard deviation operator.

- **Input rate Mean Crossing:** This feature calculates the amount of times the input rate time series restricted to the 25 most recent time steps has crossed its mean value.
  Let $\bar{X}_k = \frac{1}{25}\sum_{i=k-24}^{k} X_i$ be the mean within the last

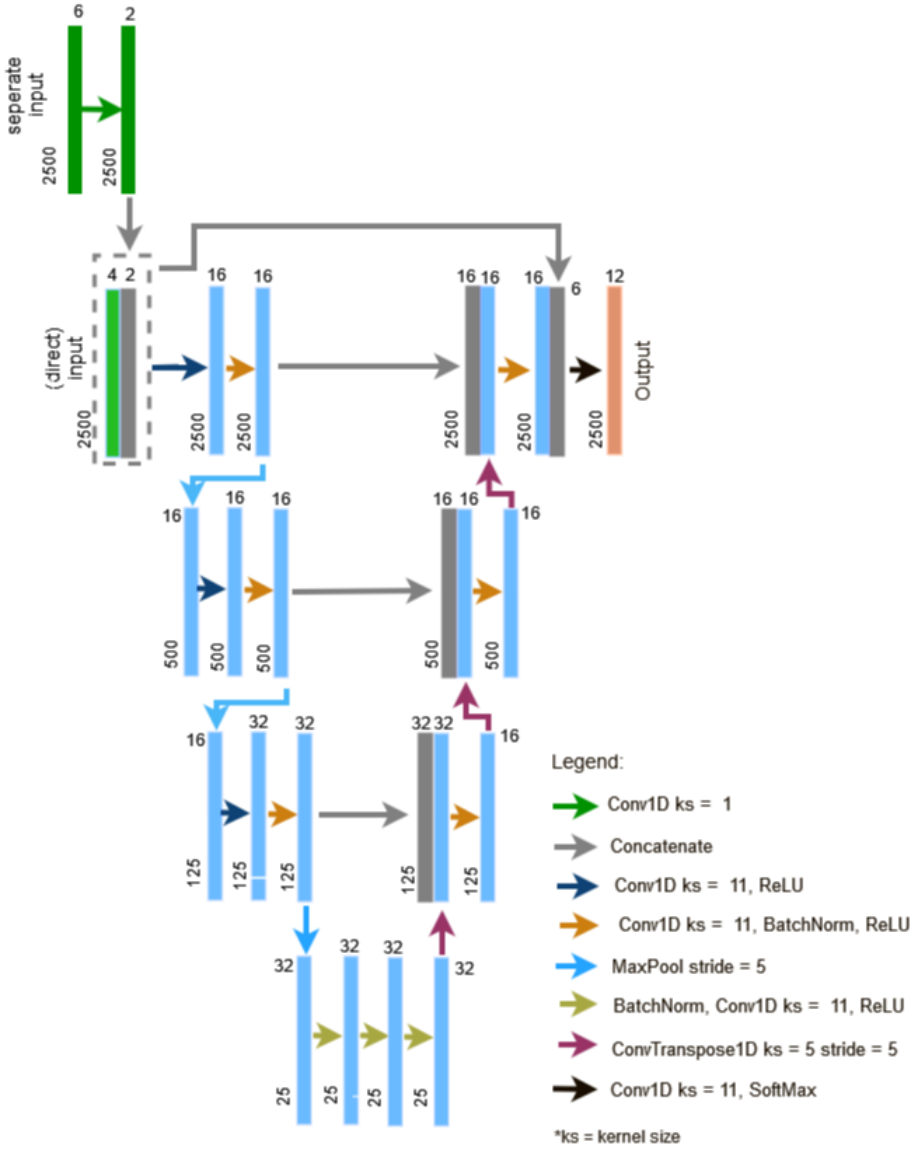

Fig. 2 – Unet Model used in the competition

25 time steps, let $g(a;x,y) := sgn(x-a)\cdot sgn(y-a)$ and define

$$C_k\left(X_{m-1}, X_m\right) = \begin{cases} 1, & \text{if} \quad g(\bar{X}_k; X_{m-1}, X_m) = -1 \\ 0, & \text{else} \end{cases}$$

Then, the new feature is calculated as follows:

$$Y_k^{mean\_crossing\_portIn} = \sum_{i=k-23}^{k} C_i(\bar{X}_k; X_{i-1}, X_i)$$

- **Input Rate Peak Difference:** This is the difference between the maximum and minimum calculated across the 25 most recent values of the input rate.

$$Y_k^{peak\_diff\_portIn} = \max\{X_i^{portIn}\}_{k-24\le i\le k} - \min\{X_i^{portIn}\}_{k-24\le i\le k}$$

- **Input Rate Time since Zero:** The amount of time passed since the input rate has been zero for the last time.

$$Y_k^{time\_since\_zero\_portIn} = k - \arg\max\{X_i^{portIn} \mid X_i^{portIn} = 0 \quad \wedge \quad i \le k\}$$

- **Cache Queue Size Time since Zero:** The amount of time passed since the cache queue size has been zero for the last time.

$$Y_k^{time\_since\_zero\_qSize} = k - \arg\max\{X_i^{qSize} \mid X_i^{qSize} = 0 \quad \wedge \quad i \le k\}$$

Let's summarize shortly where the motivation for these features comes from: Being one of the most key statistical measures in general the standard deviation is a feature

that the neural network can not easily compute itself. At the same time the input rate and cache queue show regions of high fluctuations and regions of lower fluctuations, and therefore the standard deviation might be helpful to distinguish different scenarios. The count of mean crossings is an attempt to capture the trend of the time series. It gives a hint to whether the input ratio is constantly increasing or decreasing or whether there is some oscillation between higher and lower input rates. The peak difference feature is meant to capture large jumps in the input rates. Lastly, the two features counting the time since the last zero are inspired by the observation that there seem to be sections of the time series where the input rate goes to zero frequently, while not going down to zero for a long time in other sections. The same holds true for the cache queue size, even though the cache size equals zero a lot more often in general than the input rate.

None of the features includes values of the output rate. Features calculated on the output rate have only been explored scarcely because of the high correlation between the input and output rate. It might be worth noting though that the correlation itself between input and output rate has been tested as a feature in the hope, that marking the rare occasions where output and input are not aligned might help classify the scenarios. However, the feature did not prove to be benifical to the classification accuracy in the auxiliary task.

3.2 The fourth direct input feature - mode ratio

While the six feature defined in the above subsection all improve performance, none of them improves it by as big of a margin as the feature chosen to be the fourth direct input feature. We will call it the (input rate) mode ratio. It calculates the relative amount of values among the most recent 25 time steps of the input rate time series that have been equal to their mode. Let M_k be the mode of the subsequence $\left(X_i^{portIn}\right)_{k-24\leq i\leq k}$, then we can formally define the mode ratio feature as

$$Y_k^{mode\_ratio\_portIn} = \frac{1}{25}\ |\{X_i^{portIn} \mid X_i^{portIn} = M_k \ \wedge\ k-24 \leq i \leq k\}|$$

To understand where the inspiration for this feature comes from, a close look has to be taken at the input rate time series. Figure 3 shows an example of why the mode ratio feature seems to be working so well. While it is not surprising that the mode ratio is high in regions where the input series itself becomes zero frequently, a non-zero mode ratio in regions where the input rate does not hit zero is very interesting. It seems that on some occasions the time series exhibits the exact same non-zero input rate more or less consistently for several time steps, while on other occasions the fluctuation is much more random, and the rate does not assume any fixed value at all. Importantly, these differences in the input rate behaviour seem to partially correlate to the current scenario. To support the exemplary observation made in figure 3, figure 4 shows a boxplot of the distribution of the mode ratio feature among the different scenarios of the training set. Even the superficial look at the distribution of the feature shows that it can be powerful when trying to distinguish among certain scenarios. The positive effect of the feature on the model's performance is suspected to arise from this discriminatory power.

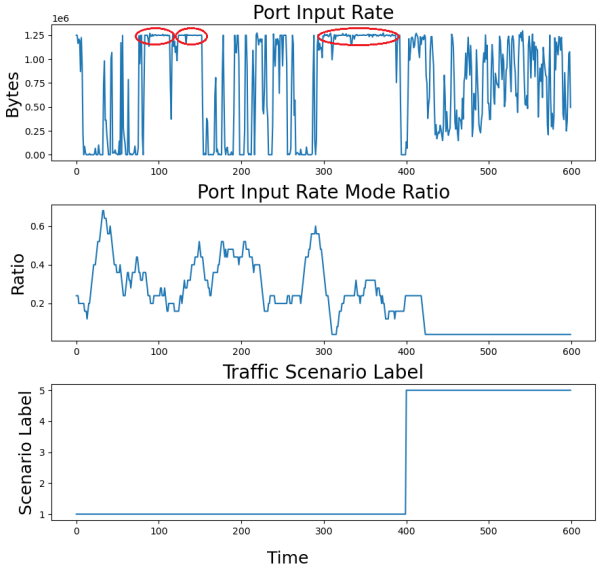

Fig. 3 – Behaviour of the mode ratio feature.
Top: Some regions of the input rate exhibit the tendency to assume the very same non-zero value multiple times within short windows of time. Some of these regions can even be distinguished by eye (red circles). *Middle:* Mode ratio feature calculated for the input rate sequence on top. *Bottom:* Corresponding scenario label. It becomes appearant that once the scenario changes from 1 to 5, the mode ratio suddenly drops to zero.

3.3 Cross-Validation of Separate Input and Mode Ratio

While the positive effect of the mode ratio feature is undeniable, the question whether the separate input features actually increase performance is less definitive. During the competition the decision was made to include the separate input in most of the models and in particular in three of the four final ensemble models. However, after the competition, another cross-validation run was done in order to further investigate the possible benefit of the hand-engineered features. A 10-fold cross validation was done on the training set, training a model following the architecture and training parameters of Model 6 (see table 2 for reference).

Table 1 reports the highest average validation accuracy, the epoch e_{max} that exhibits the highest average validation accuracy and the mean average validation accuracy among the 15 epochs before and after e_{max} to gauge the consistency in reaching peak accuracy. Here, average

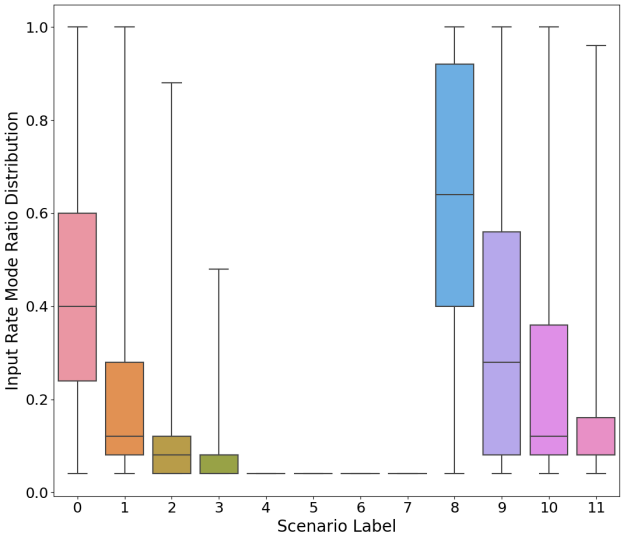

Fig. 4 – Boxplot of the mode ratio feature distribution for each class. In some scenarios the input rate almost never takes constant values (scenarios 4 - 7), while in others constant rate values are rare but existent (e.g. scenarios 2,3 and 11) or even very common (e.g. scenarios 0,8 and 9). Note that outliers are not shown in this plot as every single scenario has at least a few outliers that reach the maximum value of 1 (presumably due to zero input rates and padding).

refers to the average of the 10 cross-validated splits. All models have been trained for 300 epochs.

As the results show, the model without any additional features actually outperforms the model with only the separate input features as addition, for this particular model setup. On the other hand, the model that uses both, separate input and the mode ratio feature, shows slightly better performance on average than the model that uses only the mode ratio feature as its fourth direct input. Thus, it has to be said that the benefit of the features used as separate input, or any subset of those features, remains inconclusive and may very well depend on the particular choice of features and the specific model and training parameters. Meanwhile, the increase in performance from adding the mode ratio feature is obvious.

Table 1 – 10-fold Cross Validation performance of model with and without separate input and mode ratio feature; mean average accuracy calculated across 15 epochs before and after best epoch

Model	Peak Average Accuracy	Best Epoch	Mean Average Accuracy
No additional features	74.20%	299	73.93% ±0.14
only separate input	73.79%	262	73.46% ±0.19
only mode ratio	75.60%	251	75.34% ±0.14
separate input and mode ratio	75.58%	267	75.41% ±0.10

4. POSTPROCESSING OF MODEL PREDICTIONS

The UNET model's output is a 12-dimensional probability vector for each individual time index of the series. While the overall accuracy of these predictions (choosing the class with highest probability as the predicted class) is already high, a drop-off in quality can be observed corresponding to the predictions' relative position within the input subsequence.

Both, training and inference, are done on subsequences of input length 2500 - relatively short sequences compared to the whole time series. When looking at an input sequence centered around a particular point in time t, $x_{t-1250}, ..., x_{t+1249}$ and its corresponding model output $\hat{y}_{t-1250}, ..., \hat{y}_{t+1249}$, let us define a ratio $s \in]0, 1]$. For fixed s we are interested in the performance of the output's subsequence $\hat{Y}_t^s = \hat{y}_{t-(1250 \cdot s)}, ..., \hat{y}_{t+(1249 \cdot s)}$, that is, the portion of size s of the output that corresponds to the predictions closest to the center of the particular window of inference.

During the competition, a random 80-20 training-validation split was used to train a model and examine the accuracy of $\hat{Y}_t^s$ in dependence of s on the validation set. Figure 5 recreates the original analysis done. While the initial drop in performance followed by a steep increase is certainly interesting and might be worthy of a separate investigation itself, the main takeaway of Figure 5 with regards to reaching the best performance possible is that the accuracy of $\hat{Y}_t^s$ is maximized roughly around $s = 0.25$, and in particular, that the performance for values of s between 0.2 and 0.5 is higher than for $s = 1$. This implies that the predictive power towards the ends of the prediction window is lower. This result should not come as a surprise as the model can take more context into account the closer to the center of the prediction window the indices it predicts are located. In particular there are better ways of doing inference than simply using the model's immediate output.

Based on this insight, several options come to mind that could be explored in order to further improve the model's performance. One option is to simply increase the input length, thus reducing the amount of indices that are close to the ends of the prediction window. This is the easiest and most direct option. However, when choosing this option one should consider increasing the training input length as well, in order to keep the amount of padded values the model sees constant between training and inference. This option is certainly worth exploring, but has not been chosen during the competition. A second option might be to fix $s = 0.25$ and instead of creating just one prediction for each of the series' indices, letting the prediction window slide across the series with a step size of $2500 \cdot s = 625$. This way, four different predictions are created for each index. One of which corresponds to a position on the left end of the input sequence (i.e. the index

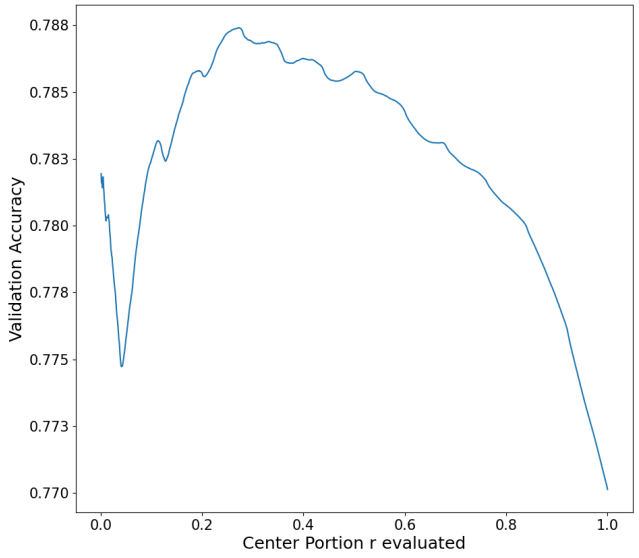

Fig. 5 – Validation accuracy as a function of the center portion s the accuracy is validated on. Accuracy is higher when using only predicitions that are close to the prediction window's center. (Note: The variable in the figure should be called s, not r.)

is among the first 625 input indices), one on the right end of the input sequence (index is among the last 625 input indices), one in the center (index is among the 625 indices closest to the center of the sequence) and a last prediction corresponding to a position 'inbetween', that is neither on the ends nor in the center. Finally, out of the four predictions created, the central prediction might be chosen as the one with the highest average accuracy.

With Figure 5 and its related analysis proving that the second option works well and it being an easy solution, it might be the preferred option in a practical setting assuming the model has been trained on an input length of size 2500. However, during the competition the set goal is to extract the maximum amount of accuracy possible. Hence, a third option was chosen for the final submission. Instead of discarding the other three predictions and only choosing the center prediction, a simple logistic regression model was fit to the four predictions in order to create a weighted average. In particular, the logistic regression model takes in 48 variables (4 vectors of 12 class probabilities) and puts out a single vector of size 12 indicating the combined class probabilities after aggregating the left, right, center and 'inbetween' predictions of the model. Based on observations during the competition the expected increase in performance from this logistic regression compared to always choosing the center prediction is consistent.

5. MODEL ENSEMBLE AND PREDICTION SMOOTHING

After postprocessing the model's output, the individual model's prediction is complete. In the competition's final submission an ensemble of four UNET models was used. The labels that the ensemble put out were additionally smoothed to avoid fast fluctuations in the predicted scenarios. The smoothing does improve performance marginally but was mainly used to achieve a class prediction that makes sense from a practical standpoint.

5.1 Four Model Ensemble

The first of the four models follows the exact architecture and training procedure described in section 2. The other three models are slight deviations from this model. Table 2 explains the modifications applied to obtain the three additional models. On top of the models using slightly different architectures and training parameters, the random seeds were chosen differently as well across the four models, in particular to feed the training examples into the models in different orders.

Table 2 – The four ensemble models

Model ID	Description
1	This is the model presented in section 2
2	Same as model 1 but using only the four direct inputs, not the separate input. The kernel size of the very last output layer is reduced to 3 instead of 11. The model was trained for 175 instead of 180 epochs.
4	Same as model 1 but using a weight decay of 0.02. Also, past epoch 40 the learning rate decreases exponentially every epoch by a factor of 0.992. The model was trained for 175 instead of 180 epochs.
6	Same as model 1 but using a weight decay of 0.04 and a lower initial learning rate of $8 \cdot 10^{-5}$. Also, past epoch 40 the learning rate decreases exponentially every epoch by a factor of 0.992. The model was trained for 200 instead of 180 epochs.

The four distinct predictions of the models are gathered to create the ensemble prediction in the same way the predictions of overlapping prediction windows were summarized (see section 4). That is, we train another logistic regression that takes the 48 variables produced by the four models (4 prediction vectors of 12 probabilities each) and outputs one single probability vector for the 12 classes. The accuracy on the hidden test set reached during the competition of the four models and the ensemble is summarized in table 3. Compared to the best performing individual models, the ensemble further improves accuracy by around 1%. Note that the submissions scored in table 3 already include the very last step of prediction smoothing. That step will be discussed in the following subsection.

5.2 Prediction Smoothing

Once the (postprocessed) predicted class probabilities of an individual model or the ensemble are created, the predicted class is then chosen to be the class with the highest

Table 3 – Test set accuracy for the four individual models and their ensemble created during competition. All submissions shown include postprocessing and smoothing.

Model	Public Test set Accuracy	Private Test set Accuracy
Model 1	77.25%	77.13%
Model 2	78.14%	78.07%
Model 4	77.56%	77.48%
Model 6	77.54%	77.41%
Ensemble	79.16%	79.05%

probability. When looking at the predicted classes, one can observe sections of the predicted time series where the predicted class changes very often within a short time interval, usually oscillating between two classes. While this behaviour is simply reflective of the model's uncertainty to decide between the two classes with the highest probability, we know from the training data that all scenarios are at least 300 time steps long, such that a fast oscillation between two scenarios does not happen in practice.

To reflect this reality within the predictions, a smoothing routine was applied to the predicted classes. Given a fixed smoothing radius $r \in \mathbb{N}$, the predicted class $\hat{y}_t$ at point t was replaced by the majority voted class of all predictions within the interval $[t-r, t+r]$, that is

$$\hat{y}_t^{smooth} = Mode\left(\{\hat{y}_k \mid k \in [t-r, t+r]\}\right)$$

This way, short fluctuations away from the locally dominant predicted class are smoothed out and the prediction appears a lot more like the sequences of scenarios measured in practice. The specific value of the smoothing radius r was set to be $r = 75$. It turns out that this routine has the secondary advantage of actually improving the accuracy slightly as well. See figure 6 for an example of predicted classes before and after smoothing and the improvement of validation accuracy when applying the smoothing routine. The results in figure 6 were obtained by training and evaluating a single UNET model on a random 80-20 training-validation split.

6. FULL MODEL PIPELINE

The full model pipeline is shown in figure 7, to summarize all different parts of the prediction process presented in this report. First, the four neural network models create their predictions using overlapping prediction windows. Each of the individual overlapping model predictions are then postprocessed and summarized by applying a logistic regression, resulting in a 12-dimensional probability vector as output for each of the individual models. The four individual predictions are then fed into a second logistic regression that creates the model ensemble prediction. The class with the highest predicted probability is chosen as the predicted class of the model. Finally, the predicted classes are smoothed by applying a local majority vote (i.e. calculating the local mode).

To conclude this report, table 4, as an addition to table 3, contains the public and private scores reached for different steps of the pipeline, i.e. when not using any postprocessing or using the center predictions instead of applying a logistic regression. It is added to give an idea of the importance of the different steps of the pipeline in terms of improving performance.

7. OUTLOOK

The model presented in this report shows that 1D UNETs are a promising approach to the problem of network traffic scenario classification. Additionally, some other valuable insights were gained, such as the strength of the mode ratio feature and the usage of a smoothing routine to make predictions more stable. At the same time, there are some aspects of the solution that still need further inspection. Topics that might be worth to look into in further analyses of the problem include:

- **Hyperparameter Tuning**: While some hyperparameter tuning was done, the proposed model can surely still be improved upon with a more in-depth investigation of the network architecture and its parameters (i.e. number of blocks and layers, regularization ...)

- **Hand-engineered Features**: While the mode ratio feature proved to be very valuable, the analysis of whether the remaining hand-engineered features that were used in the final model are actually beneficial or not has not been fully conclusive.

- **Overlapping Window Predictions and Input Length:** The idea of generating four different predictions for each index according to a overlapping sliding window approach and summarize those predictions in an additional logistic regression proved to enhance performance for the particular model at hand. However, it is unclear whether the same or even superior performance might be reached by significantly increasing the input length of the subsequences fed into the model.

- **Real-Time Predictions:** The model makes use of 'future' features, that is, to predict the scenario at time t, features are included that were measured at times $t+k$ with $k > 0$. For real-time prediction applications these features are not available. Thus, the problem of turning the model into a real-time prediction model, and investigating the drop-off in performance that will be inevitable when doing so, might be very interesting from a practical point of view.

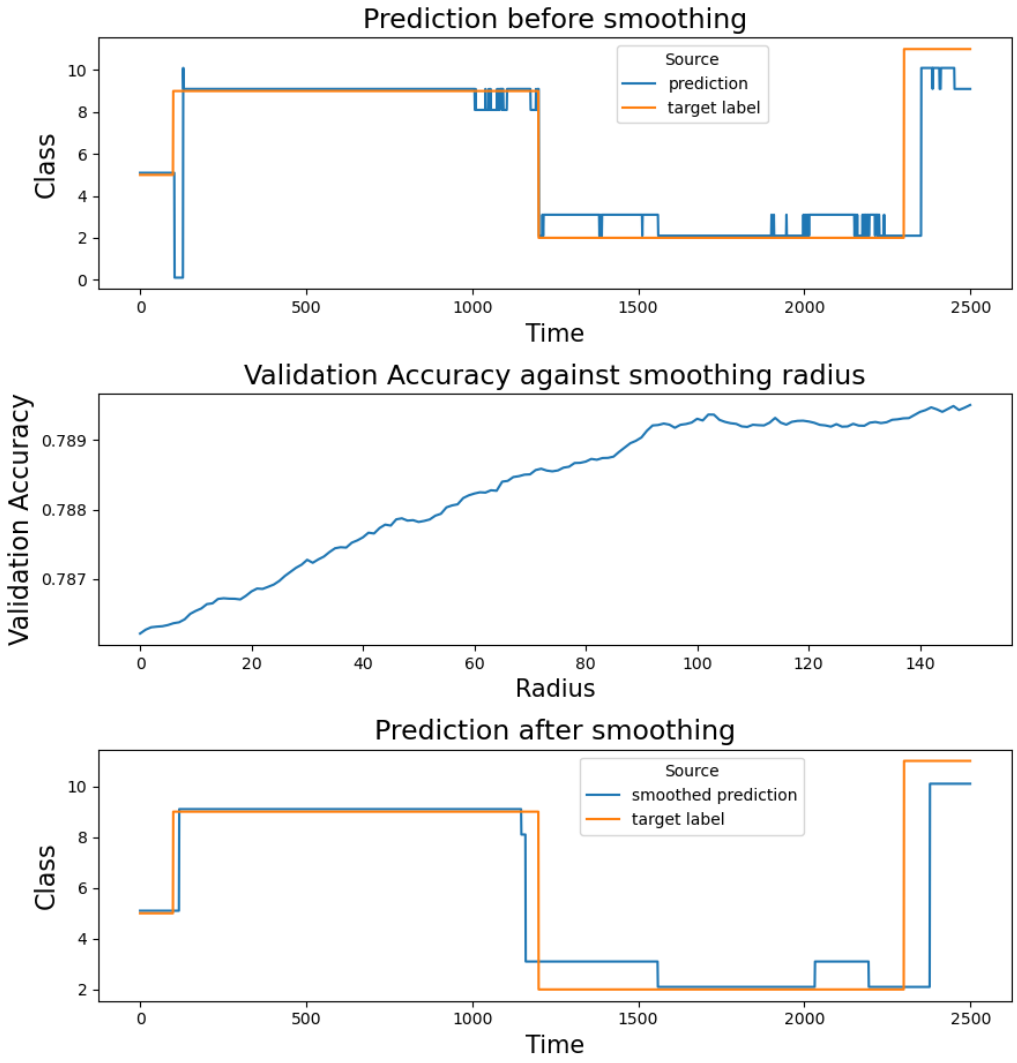

Fig. 6 – Effects of smoothing the predicted class.
Top: Predicted classes before smoothing.
Middle: Accuracy increases when using the smoothing routine.
Bottom: Predicted classes after smoothing with $r = 75$. The predictions have a much closer resemblance to an actual sequence of scenarios.

Table 4 – Test set accuracy for the four individual models and their ensemble when stoping at different steps of the pipeline; Accuracies are reported as [public score] / [private score]

Model	No postprocessing	center prediction and smoothing	full postprocessing	ensemble without smoothing	final ensemble
Model 1	74.86% / 74.38%	76.75% / 76.64%	77.25% / 77.13%	-	-
Model 2	75.23% / 75.20%	77.54% / 77.44%	78.14% / 78.07%	-	-
Model 4	74.90% / 74.81%	76.93% / 76.87%	77.56% / 77.48%	-	-
Model 6	75.16% / 75.07%	76.93% / 76.85%	77.54% / 77.41%	-	-
Ensemble	-	-	-	78.96% / 78.87%	79.16% / 79.05%

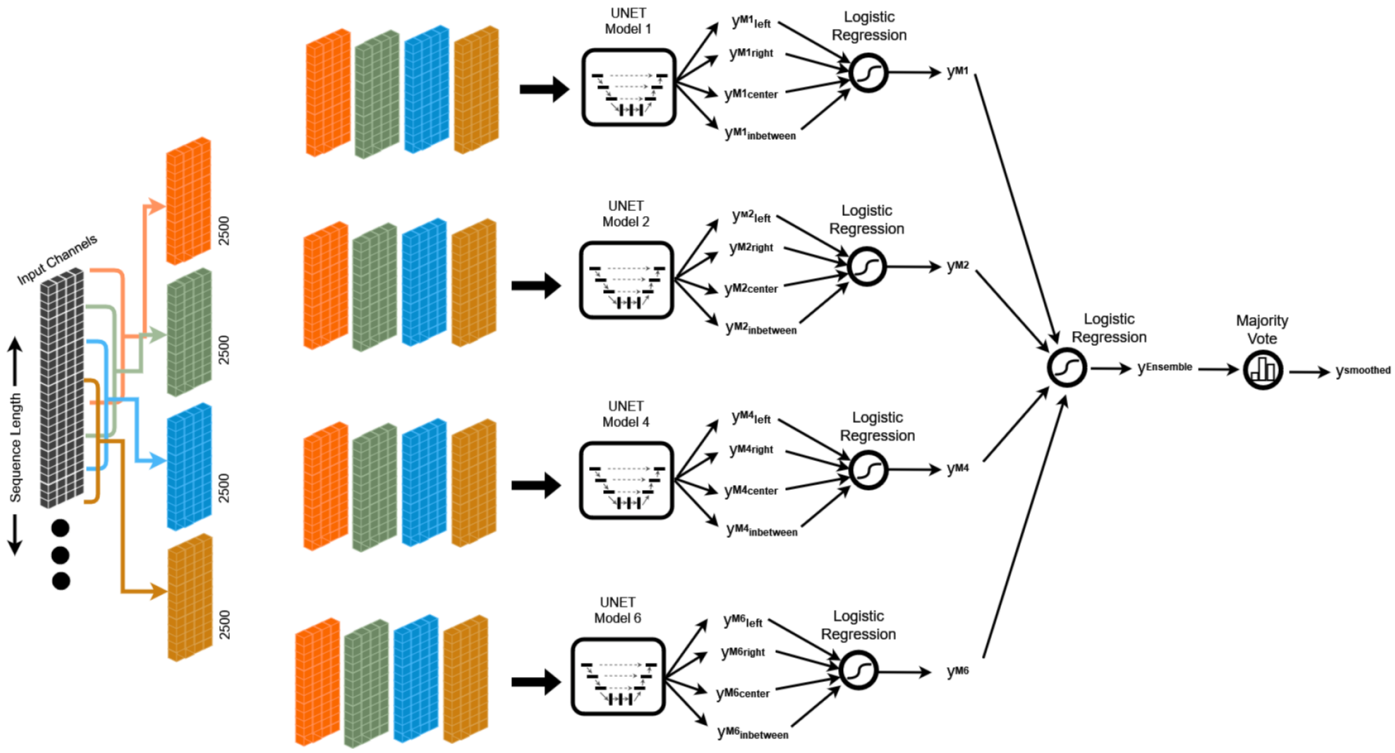

Fig. 7 – Full model pipeline. The input series is cut into overlapping prediction windows of size 2500 that are fed into each of the four models. The four models put out four probability vectors corresponding to each of the four positions of the indices relative to the prediction window. The four probability vectors are then summarized in a logistic regression to form the final individual model prediction. All four final model predictions are then again summarized in a second logistic regression to form the ensemble probability vector. The predicted class is chosen to be the class with the highest probability. The final output is created by smoothing the predicted classes using a local majority vote.