Classification of Network Traffic Using Flow-Based Features and Deep Learning

Jaan Van Gils (6200443), Maarten Al Sadawi (6061559), Tim Smit (6527906),
Silvester Graas (4175778), Willem van Veluw (6278957), Martijn Jansen (6513328)

Abstract—The classification of network traffic is of considerable importance to parties involved with Quality-of-Service (QoS) provisioning and security. Due to the increasing amount of network traffic, as well as the fast evolving nature of the traffic, the field is under a constant pressure to adapt. Currently, deep learning methods are the state of the art in network traffic classification. In this work a multi-task Recurrent Neural Network (RNN) and a Convolutional Neural Network are presented. The performance of the CNN multi-task network is compared to that of of two parallel single task networks. The models are trained using flow-based features and predict the traffic type, as well as the application that generated the data. The multi-task CNN outperforms the multi-task GRU, as well as two single task CNNs. The multi-task CNN scores a F_1 of 95.7% and 96.1% for traffic type and application respectively.

1 Introduction

Network traffic classification is an important topic of research for many parties that are concerned with Quality-of-Service (QoS) provisioning, pricing, malware detection and network security [17, 1]. Therefore, many different techniques have been explored and developed over the years to serve this purpose [24]. Due to the constantly evolving nature and increasing volume of network traffic there is a constant drive for innovation in the field [1]. Some examples include the categorisation of traffic generated by Zero-day applications and the growth of big data and IoT [21, 19]. Techniques for network traffic classification can be broadly subdivided into three approaches [2]. The first approach is based on port-inspection, i.e. the direct assignment packet port-number of a network packet to its service/application [16]. However, new techniques like random-port assignment and tunnelling make this approach unusable in many cases [20]. The second technique involves inspection of individual network packets. This can be subdivided into shallow packet inspection (SPI) and deep packet inspection (DPI). SPI only inspects the header of the packet; DPI inspects the complete packet, including the payload [9, 22]. Although DPI methods give more information regarding the packet it also suffers from more drawbacks. These include privacy concerns, the inability to be applied on network traffic captured from a Virtual Private Network (VPN) or encrypted traffic and the high computational overhead of the payload inspection [20, 12, 17]. The third approach in network traffic classification is the observation of statistical (time-based) features of a complete network flow [8, 17, 1]. A flow can be defined as 'a sequence of packets sent from a particular source to a particular unicast, anycast, or multicast destination' [4]. flow-based features have proven to be very useful as they can also be used to categorise VPN/encrypted network traffic and do not depend on payload inspection. The use of flow-based features relies on the assumption that different services/applications generate a unique fingerprint of statistical (time-based) features. Some examples of these features are flow duration, the number of packets sent forward/backward (per second) or the mean time between two packets sent in a flow.

Network Traffic classification can be performed to predict labelling related to the final objective of the classification. Some possible labels include encryption type (VPN vs. non-VPN) [8], the used communication protocol (e.g., FTP, HTTP, UDP), browser-type (Chrome, Firefox), operating system (e.g., Android, Windows), traffic-type (e.g., streaming, VoIP, chat) or application type (e.g., YouTube, Netflix, Skype).

Although deep learning methods have been around for longer, their use in the network traffic classification task have only been of the last few years [17]. Deep learning approaches, contrary to traditional machine learning methods, have the property that their performance generally increases with the amount of data given. Also, deep learning techniques are less dependent on knowledge of domain experts. With the fast evolving nature of network traffic as well as the increasing volumes of internet traffic, deep learning becomes a very suitable and appealing option for network characterisation [1]. Following a suggestion by [17] this research will explore the performance of multi-task learning on the classification of two related network traffic labels. In multi-task learning, a problem that consists of multiple related sub-problems is learned at once using a shared representation [5, 13, 7]. The general idea is that the related tasks in question both share low-level representations, which are better learned when taking both tasks in consideration. Especially when there are strong correlations between the labels of each task, the optimisation of the model to one task benefits the prediction on the second task. Multi-task learning model also benefit from a decreased risk on overfitting and increased data efficiency [13]. In this study the performance of a multi-task model predicting (1) traffic type and (2) application will be compared to two single task models that predict the two labels in parallel. The models will be trained on flow based features. To aid our discussion, the following two research questions will be answered:

(**RQ1**): *How does the flow-based Convolutional Neural Network compare to the Recurrent Neural Network when predicting both*

traffic type and application type simultaneously?
(**RQ2**): *How does the highest performing multi-task model compare to a model predicting only traffic type and a model predicting only the application?*

2 Related Work

Table 1 summarises the main approaches and goals of the related work. The table compares the different studies on the used dataset, the type of features used, the model(s) employed and the final goal of the research. As can be seen quickly from the table, none of the studies combines the use of flow-based features and a CNN/RNN deep learning architecture in order to classify traffic type *and* application type. In 2016 [8] investigated classification of traffic types of traffic flows as well as classifying whether the flow is VPN or Non-VPN traffic. The authors of this work also created the ISCX-VPN-NonVPN dataset which has become the foundation of many papers in later years. Essential is the tool that comes with the dataset to extract time-based, or flow-based, features. The dataset will be considered in more detail in section 3, since it will be the dataset used in this research. With the flow-based features constructed, [8] trains two machine learning approaches: C4.5 (decision tree) and KNN (k-nearest neighbours). Although the type of the used features is equivalent to our research, deep learning methods are applied in our research. The accuracy levels of the two models used in [8] were above 80%. In [12], ISCX-VPN-NonVPN also formed the basis of the research. Although the data was used to train deep learners, it should be noted that the approach of this research is different. However, the research of [12] uses packet features as opposed to flow-based features as in this paper. As a measure of performance, it was stated that the recall was equal to 98% and 94% in application type respectively traffic type.
The goal of [23] was to construct an end-to-end method to classify (just) traffic type. The method was implemented with both an one-dimensional as well as a two-dimensional CNN, but it was again based on packet features from the ISCX-VPN-NonVPN dataset. Its goal was also to classify just traffic type, whereas this research also classifies the application. It was stated that the accuracy was higher than 80% for the two-dimensional CNN, while the accuracy for the one-dimensional CNN yielded 2.51% more. The dataset ISCX-VPN-NonVPN also appears in the work of [10], but again packet features were used and the goal was to classify just traffic type. It was stated that the baseline multi-layer perceptron and CNN outperformed the approach in [10], which is interpreted as an accuracy of somewhere below 80%. One of the most relevant papers using a different dataset is [11]. Besides the use of a CNN, this is by our knowledge the only paper also using a RNN for the classifying task. However, the model was trained on packet based features in contrary to flow based features in this work. High performance metrics (an accuracy of above 96%) were achieved by the RNN. The CNN acquired a accuracy above 95%. The performance of the RNN model was improved, however, by combining it with the CNN model. In [6] an online classifier was constructed, designed to quickly classify network traffic. Especially for online classification for security purposes, speed is essential. In [15] a combination of packet and flow-based features was used. The goal of the paper was to classify VPN or Non-VPN traffic, which is different from our research goal, but combining both types of feature will lead to interesting comparisons of model performances. The multi-task learning approach has not been studied extensively yet in the field of network traffic characterisation. However, there are some interesting examples where good results were obtained. In [18] a multi-task approach is taken to predict bandwidth requirement, the duration of a flow and traffic class. [3] applied a multi-task approach with the specific task of Intrusion Detection. Most interesting, both [18, 3] report as one benefit of the multi-task learning approach the good accuracy on the prediction of network traffic type despite having a small labelled training set. [18] explains this result by noting that the duration and bandwidth requirement are easily predicted from the flow and subsequently help in predicting network traffic type. This result demonstrates very well how one can benefit from the shared representation in multi-task learning.

3 Approach

3.1 The Dataset

In this study the ISCX-VPN-NonVPN dataset generated by the Canadian Institute for Cybersecurity (CIC) is used [8]. The dataset was generated by two fictional internet users performing tasks on common applications in a way that is intended to reflect real-world use and capturing the network traffic during these sessions. Obtaining a representative dataset is a very challenging aspect in network classification. Although the ISCX-VPN-NonVPN dataset will be of sufficient quality for research purposes, it might have limited value when used for practical model implementations. This limitation could arise as the data is generated by a limited number of users and because of the predictable nature of network traffic generated in an artificial setting. The ISCX-VPN-NonVPN dataset provides network traffic captured in a VPN and in a non-VPN setting. For both of these classes, the data is labelled according to traffic-type and application; the labels are shown in table 2. To limit the scope of this study, the binary VPN label is not predicted but taken from the data. The reason for this is that prediction of this label has been successfully demonstrated to be possible by [8].

3.2 Data Pre-Processing

In this section the pre-processing steps that have been taken in order to create the input on which the models train will be discussed. The basis of the pre-processing is a tool that comes along with the ISCX-VPN-NonVPN dataset [8]. Before focusing on the role of the tool, the notion of flows will be presented. When two users are communicating over the network, packets are exchanged bidirectionally. This sending of packets is not a continuous stream: between every packet sent there could be a small or large time interval. Essential to the notion of flow is the assumption that the flow has stopped if this time interval is ”too large”. This assumption is especially relevant for UPD traffic as, opposed to TCP, traffic packets using UDP do not contain a flag

TABLE 1
Overview of related papers. Here *DL* and *ML* stand for deep learning respectively machine learning.

Paper	Year	Database	Features	Model	Goal
Draper-Gil et al. [8]	2016	ISCX-VPN-NonVPN	Flow based	C4.5, KNN	VPN/Non-VPN, Traffic type
Lopez et al. [11]	2017	RedIRIS	Packet	CNN, RNN	Application type
Lotfollahi et al. [12]	2017	ISCX-VPN-NonVPN	Packet	SAE, CNN	Traffic type, Application type
Wang et al. [23]	2017	ISCX-VPN-NonVPN	Packet	CNN	Traffic type
Chen et al. [6]	2017	Not referred	Flow-based	Online CNN	Protocol, Application
Miller et al. [15]	2018	Created own	Packet and Flow-based	MLP	VPN/Non-VPN
Iliyasu et al. [10]	2020	ISCX-VPN-NonVPN	Packet	Convolutional GAN	Traffic type

TABLE 2
The classes given by the ISCX-VPN-NonVPN dataset, adapted from [Draper-Gil, 2016]

Traffic-type	**Application**
Web browsing	Firefox and Chrome
Email	SMPT/S, POP3/SSL and IMAP/SSL
Chat	ICQ, AIM, Skype, Facebook and Hangouts
Streaming	Vimeo and Youtube
File Transfer	Skype, FTPS and SFTP
Video	Facebook, Skype, Hangouts
VoIP	Facebook, Skype and Hangouts (1h)
P2P	uTorrent and Transmission (Bittorrent)

to indicate that the stream has finished. The aforementioned assumption is used by the tool that came along with the ISCX-VPN-NonVPN dataset [8]. The *flow time-out* parameter is essential in this tool, since it specifies the "too large" time interval between two packets. Note that for increasing values of *flow time-out*, less flows are extracted from ISCX-VPN-NonVPN.

Since the input of the networks has to be a sequence, especially for the RNN, the tool is used twice. First the *flow time-out* is chosen as large as possible (120 seconds) in order to generate flows that will be denoted as superflows. Then the tool is run a second time, with the *flow time-out* set to 3 seconds, in order to generate flows that will be denoted as miniflows. Note that because these flows are generated using the same packets it is possible to link miniflows to the superflows that contain them. Whenever a miniflow has the same source IP and port, same destination IP and port, and has a start time and duration that fall within that of a superflow, this miniflow can be coupled to the superflow. Note that this is a unique combination, as a given port on a user system can be allocated only once at a given time. Because of this, the superflow as well as all of the miniflows have the same labels. To easily extract the labels later on, the files are renamed using some specific rules[1]. Next, the miniflows in a superflow are put into sequences of length 20. This happens as follows: first, miniflows 1 through 20 are put into a sequence. Then miniflows 2 through 21 are put into another sequence. This repeats until the end of a superflow is reached. Hence, if a superflow consists of n miniflows, a total of $\max\{n - 19, 0\}$ sequences of miniflows is generated for that superflow. Proceeding the pre-processing in this fashion yielded 10,908 sequences in total.

Next, the sequences are updated such that the miniflows in them contain only the required features such as flow duration and statistics regarding packet inter-arrival time. Additionally, the labels corresponding to the miniflows are appended in one-hot encoding. Moreover, a new feature is added that indicates whether this flow comes from the VPN encrypted traffic or not. Finally, to make pre-processing easier, all miniflows that belong to a sequence are put into one csv row, as this eases the process of splitting the train, test and validation sets. For the splitting of data, the function `train_test_split` of the `sklearn.preprocessing` package in Python is used. It is invoked with `random_state` set to 42, and `stratify` set to the two labels (that is, the combination of application name and application type). This `stratify` is required to make sure that all sets receive an almost equivalent distribution of labels, therefore removing the data bias in these sets. The split used is a 70 / 15 / 15 split for the training set, validation set and testing set respectively. It is important to note that all (built-in) random generators have been given a seed of 42.

3.3 Distribution of Labels

In this section the distribution of the labels of the sequences that form the input of the research will be discussed.

The first note is that the dataset containing the sequences is imbalanced. For instance, there are many more *VoIP* sequences generated than *P2P* sequences: respectively 6569 and 19. It is also the case that by generating the sequences, some labels of the initial ICSX-VPN-NonVPN dataset disappeared. For example, there were no sequences with the label *Web Browsing* as traffic type. The second note is that there is a dependence between the two tasks. This can be read from table 9 in Appendix B. In this table, applications that do not occur in combination with any traffic type (i.e. rows which would have the value 0 for every column) are not shown. Therefore, only 16 applications are displayed, although the original dataset contains 21 labels. The last column of this table contains the proportions of application types among the sequences, while the body of the table contains proportions of application types given the traffic type. For example, it can be seen that 17.7% of the sequences having the *Chat* label also has the *Hangouts* label. Comparing the proportions in the body of the table and the last column shows the dependence. Indeed, when the tasks are independent of each other, the proportions in the body should resemble the ones in the last column. This is not the case in Table 9. For instance, the proportion of sequences with *Vimeo* as application equals 0.050, while the proportion of sequences with *Vimeo* as application among the sequences with *Streaming* traffic type is 0.644. The latter

1. These rules can be found in https://github.com/TKForgeron/INFOMPR-Project/blob/master/preprocessing/csv_rename.py

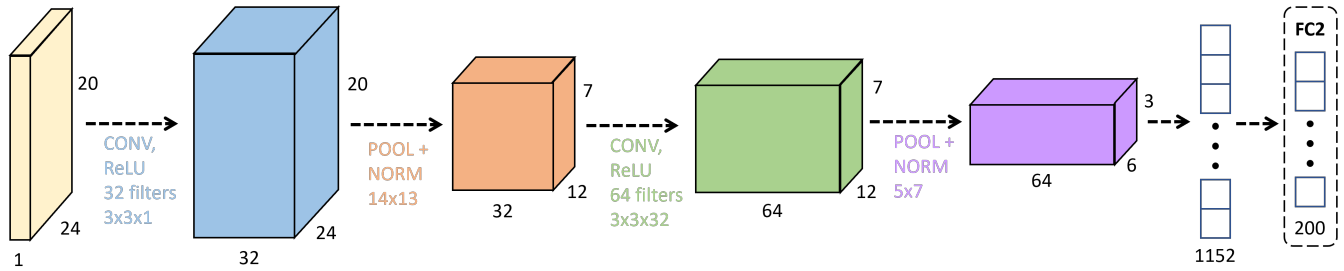

Fig. 1. Architecture of the CNN base.

is much higher than the former proportion, indicating a strong association of the tasks. Similar conclusions can be drawn by investigating other examples.

3.4 Neural Network Platform

All of the networks described in sections 3.5, 3.6 and 3.8 are set up in Python, using the `keras` module of the `tensorflow` platform [2]. The models will be set up using the Functional API, as this allows for creating multi-output models. As this API also allows for single-output models, the same API will be used for the single-task model as this will allow for code reuse. The layer APIs used are: `Input`, `Reshape`, `Dense`, `Softmax`, `Conv2D`, `BatchNormalization`, `MaxPooling2D` and `GRU`. Note that all parameters are set to the default of `keras` 2.7.0 unless otherwise specified in the following sections. The neural networks were run on a laptop with an Intel Core i7-8565u cpu, 12 GB of ram and a GTX 1650 with Max-q design.

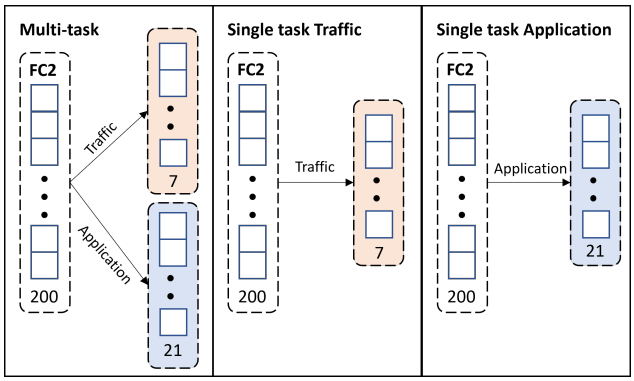

Fig. 2. Architecture of the CNN tails.

3.5 CNN Architecture

The architecture of a CNN consists of two parts: in the first part the images are processed and the features are learned, in the second part the image is flattened and classified. The architecture of the multi-task CNN is presented in the same two parts. The model was run for 100 epochs and with a learning rate of 0.00004, in order to avoid overfitting.
The first part of the model, referred to as the "base of the model" will be described first. This base is also illustrated in Figure 1. First note that a miniflow has 24 features. As a sequence contains 20 miniflows, the total amount of features in one sequence equals $20 \cdot 24 = 480$. Therefore, the input layer of the network has a shape of 1×480[3]. Because these features need to be represented as a 2D image, the input layer feeds directly into a reshape layer that creates a 20×24 shape. This means that every row represents a miniflow, and every column represents a feature.

Next, a convolutional layer with a number of filters and window size that will be tuned and a stride of one is added. After this layer, a max pooling layer is added, also with a window size that will be tuned. Additionally a batch normalisation layer is added, as this has proven to be a good addition according to [11]. These last three layers are then repeated with different tunings. Next, the output will be flattened using another reshape layer. Hereafter a fully connected layer of 200 units is added, also in correspondence with [11]. This layer is called "FC2" in Figure 1. From this point the network will branch into two separate, yet very similar parts. This is illustrated in Figure 2. Both have a fully connected layer of the same size as the labels to be output by the branch, as well as a softmax layer to return a probability distribution of these labels. Note that the weight adjustments in the part before the branch in the training phase will be based on the loss due to both the traffic-type and the application outputs. This will be elaborated on in section 3.7.

3.6 RNN Architecture

The RNN was also run for 100 epochs. The learning rate used is 0.0008. The first layer of the RNN is the input layer. Like the CNN, it has a shape of 1×480 as well. It will also be reshaped in the second layer, to a 20×24 shape. In this reshaped layer, the 20 rows reflect the 20 flows in the sequence and correspond to the red boxes in Figure 3. Then follows a Gated Recurrent Unit (GRU) which outputs a layer of size 24, referred to as "FC1" in Figure 3. The parameters have been chosen such that the layer is compatible with CuDNNGRU. This means that tanh is used as activation function and sigmoid as reccurent activation function. At this point, a branch is added to the network that creates two similar blocks for both labels. Both of the branches will have a fully connected layer, that connects layer "FC1". This fully connected layer is then fed into a softmax layer, to receive a probability distribution of the labels.

2. See https://github.com/TKForgeron/INFOMPR-Project for the complete implementation.

3. See section 3.2 for why all features are in a single vector.

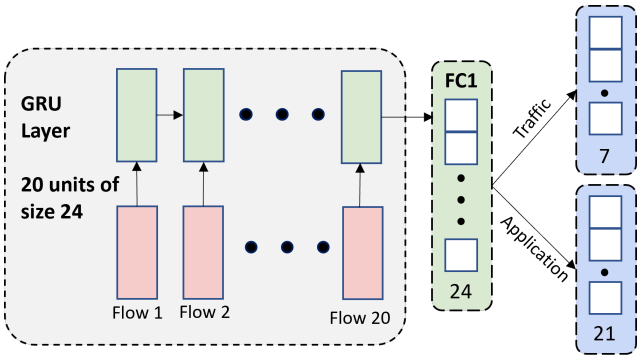

Fig. 3. Architecture of the RNN.

3.7 Multi-Task Loss

As the multi-task models have two outputs, one for each label, there are two losses computed. In order to update the weights in the part of the model before the branch, a metric of the form $L(\vec{\theta}) = c_1 \cdot L_t(\vec{\theta}) + c_2 \cdot L_a(\vec{\theta})$ is required to compute the "overall" loss. Note that L_t and L_a are the loss metrics for traffic type and application respectively. The coefficients c_1 and c_2 indicate which of the two separate metrics should have a larger influence on the overall loss, and thus on how the model weights are updated. As the correctness of both labels is equally important (i.e. the research does not place emphasis on one of them), the values for the coefficients are chosen as $c_1 = c_2 = 1$.

3.8 Single-Task CNN Architectures

The two architectures for the single-task are very similar to that of its multi-task counterpart. The single task models both contain the CNN base, see Figure 1. However, instead of branching into two directions after layer "FC2", only one branch is added. See Figure 2 to get an overview of the follow-up layers of "FC2". Its learning rates have also been tuned down to 0.00002, to avoid overfitting.

4 Results

This section summarises the results of this research. First, research question one, which questioned the comparison between the CNN and the RNN, will be discussed. Whichever of these two models performs best will be chosen to answer the second research question, which is a comparison between the best multi-task model and two single task models. To make any statements about significant difference in performance, the McNemar test is applied with level of significance of $\alpha = 0.05$. The contingency tables used to perform the McNemar tests can be found in Appendix A. These tables already provide some insight into which models perform better. The (correct, correct) and the (incorrect, incorrect) cells are all the examples that are predicted the same by both models, while the other two cells indicate where the models differ. For the McNemar tests, the null hypothesis states that the probability $p_{(correct,incorrect)}$ is the same as the probability $p_{(incorrect,correct)}$ [14]. This means the null and alternative hypotheses for all tests are defined as follows: $H_0 : p_{(correct,incorrect)} = p_{(incorrect,correct)}$ and $H_1 : p_{(correct,incorrect)} \neq p_{(incorrect,correct)}$.

TABLE 3
Performance metrics of the models. MT stands for multi-task, ST stands for single-task, App. stands for application and Type is short for traffic type. Note that the single task model are actually two sequential models.

Model	Label	Accuracy	Precision	Recall	F_1
RNN MT	Type	0.948	0.949	0.948	0.948
	App.	0.940	0.947	0.938	0.942
CNN MT	Type	0.954	0.954	0.954	0.957
	App.	0.961	0.962	0.960	0.961
CNN ST	Type	0.953	0.953	0.952	0.952
	App.	0.949	0.956	0.944	0.950

TABLE 4
Runtime of the multi-task (MT) and the single task (ST) models in seconds. The runtime of the single task models are summed.

	RNN MT	CNN MT	CNN ST
Runtime	276.7969	222.9688	242.8906

4.1 Multi-Task CNN Compared to Multi-Task RNN

Table 3 shows the accuracy, precision, recall and F_1 score of the multi-task CNN and multi-task RNN for predicting the two labels. When looking at how well both models predicted type, the CNN outperforms the RNN in all four metrics. This is also confirmed by table 5, where the CNN model predicts nine cases more correct where the RNN model predicts them incorrect compared to the other way around. When applying the McNemar test on table 5, however, a p-value of 0.4262 is returned. As the p-value is lower than $\alpha = 0.05$, the alternative hypothesis will be rejected and the null hypothesis will be accepted. This means there is no significant difference between the multi-task CNN and multi-task RNN models when predicting traffic type.

When looking at the application label, the CNN model also outperforms the RNN model in the four metric provided in table 3. Table 6 confirms this, as the CNN model predicts 34 cases more correct where the RNN model predicts them incorrect compared to the other way around. When applying the McNemar test, a p-value of 0.0004 is returned. This is quite a bit smaller than $\alpha = 0.05$, which means the null hypothesis will be rejected and the alternative hypothesis will be accepted. This means that the multi-task CNN model predicts the application significantly better than the multi-task RNN model. In addition to this, the CNN model also has a lower runtime, being more than 50 seconds faster as can be seen in table 4.

Overall, the CNN model outperforms the RNN model in all metrics shown. It even outperforms the RNN significantly when predicting application. Because of these results, the second research question will be answered using the CNN model.

4.2 Multi-Task CNN Compared to Two Single-Task CNNs

Table 3 shows the accuracy, precision, recall and F_1 score of the multi-task CNN and the single-task CNNs for predicting the two labels. When looking at the traffic type, the multi-task model performs slightly better than the single-task model. The differences are very marginal however. This is confirmed by looking at table 7, where the CNN multi-task model only predicts one more case correct where the

single-task model predicts incorrect compared to the other way around. When doing a McNemar test on the table, a p-value of 1 is returned. This is higher than $\alpha = 0.05$, which means the alternative hypothesis will be rejected and the null hypothesis will be accepted. It seems that there is almost no difference at all between the multi-task and single-task models when predicting traffic type.

When comparing the metrics in table 3 between the two models for the application label, the multi-task model outperforms the single task model again. This is confirmed when looking at table 8. Here the multi-task model predicts 18 cases more correct that the single-task model predicts incorrect than the other way around. This results in a p-value of 0.0222, which is higher than $\alpha = 0.05$. This means that the null hypothesis will be rejected and the alternative hypothesis will be accepted. It can be concluded that the multi-task model predicts the application label significantly better than the single-task model.

To conclude, the CNN multi-task model outperforms the single-task model in all metrics shown. The multi-task model improves significantly when compared to the single task model when predicting the application label. The multi-tasks model runtime is also around 20 seconds lower, as shown in table 4.

5 Conclusion

In this paper, research questions RQ1 and RQ2 have been discussed. To answer the first question, a CNN and a RNN using GRU were set up. These networks were then tuned on the same train, validation and test data, to find the best performing multi-task network. The CNN performs significantly better when predicting the application of network traffic. Previous research by Lopez-Martin et al. on the network classification problem concluded that converting data to a time-series of sequences allows a CNN model to perform similarly to a RNN model [11]. In the research by Lopez-Martin et al., although the RNN outperformed the CNN, it was found that the CNN could come close in accuracy. A difference with this research, however, is that packet based features were used instead of flow-based features. A reason for this performance difference could be that a CNN can capture data in neighbourhoods more robustly, as during the convolution it will look at all data in a window at the same time. Concretely this means that a CNN can look at multiple (neighbouring) features in multiple (neighbouring) sequences at the same time. The CNN might be able to benefit from combining the statistical flow data in a way that the RNN cannot. In addition to this, the CNN also has a faster runtime.

To answer the second research question, the multi-task CNN was compared to two single-task CNNs that both predict one of the multi-task labels. After running the single-task CNNs on the same train and test data as multi-task CNN, it was found that the multi-task CNN is significantly better at predicting the application. It was also found that it, like with the RNN, is not significantly better at predicting the traffic type. As was shown in previous research, multi-task learning can allow one task to help learn another task [5]. Traffic types correlates with application, i.e. when one knows the traffic type the options for application is reduced significantly. One possible reason why predicting application type using the multi-task model performs better than the single task model, while traffic type does not, might be that application does not really add a lot of information for traffic type. A few applications do have a few different traffic types, these are primarily communication applications like Skype. The traffic options that come with Skype however might already be easily distinguishable from the features that were given. For example video calling requires a lot more data than a simple chat message and when having a chat conversation there can be quite some non-active time while a video call probably has no time non-active. Traffic type however, might add information to application due to it actually reducing the list of possible applications. For example, streaming might be similar to video calling in the amount of data being used and how much active time there is. If the model knows however that the traffic type is video calling, the streaming application falls away. What might strengthen this relation is the fact that there are already a lot of different traffic types to choose from. This might result in a reduced variance of how well these traffic types can be predicted when compared to application.

6 Future Research

The pre-processing steps in this research filtered out a lot of the data in the ICSX-VPN-NonVPN dataset. Only superflows longer than 60 seconds (20 miniflows of 3 seconds) were considered, which meant that only 10,908 examples remained in the dataset. As a result, the label distribution of the sequences was imbalanced and heavily dependent: some traffic type labels were present with a low proportion while other occurred with a count of 300 times higher. Also, some traffic type labels occurred only in combination with one application, e.g. *P2P* and *Bittorrent*. This could be solved in two ways in future research. Possibly the most obvious way would be to zero pad all flows which were too short. This would increase the amount of examples by a lot. Zero padding was tried for this research, however it resulted in an overflow of zeroes for both the CNN and RNN models. Another way would be to simply increase the amount of 60 seconds long flows to provide more data. The RNN model had a few parameters, most importantly the activation functions, set such that they would work with CuDNNGRU, which allowed gpu support. Due to hardware limitations, tuning without this gpu support would have taken a lot longer. However, using different activation functions could have increased the RNN models performance. In [11] it was concluded that a combination of a CNN and a RNN performed better than the two networks individually. It would therefore be interesting to see if this is also the case in a multi-task setting. Moreover it would be interesting to investigate whether the conclusion of Lopez-Martin et al. extends to flow-based features, as their research was based on packet features.

References

[1] Mahmoud Abbasi, Amin Shahraki, and Amir Taherkordi. “Deep learning for network traffic monitoring and analysis (ntma): A survey”. In: *Computer Communications* (2021).

[2] Giuseppe Aceto et al. "Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges". In: *IEEE Transactions on Network and Service Management* 16.2 (2019), pp. 445–458.

[3] Reem Aljoufi and Aboubaker Lasebae. "Multi-task Learning for Intrusion Detection and Analysis of Computer Network Traffic". In: *E3S Web of Conferences*. Vol. 229. EDP Sciences. 2021, p. 01057.

[4] Shane Amante et al. *IPv6 Flow Label Specification*. RFC 6437. Nov. 2011. DOI: 10.17487/RFC6437. URL: https://rfc-editor.org/rfc/rfc6437.txt.

[5] Rich Caruana. "Multitask learning". In: *Machine learning* 28.1 (1997), pp. 41–75.

[6] Zhitang Chen et al. "Seq2img: A sequence-to-image based approach towards ip traffic classification using convolutional neural networks". In: *2017 IEEE International conference on big data (big data)*. IEEE. 2017, pp. 1271–1276.

[7] Michael Crawshaw. "Multi-task learning with deep neural networks: A survey". In: *arXiv preprint arXiv:2009.09796* (2020).

[8] Gerard Draper-Gil et al. "Characterization of Encrypted and VPN Traffic using Time-related Features". In: *Proceedings of the 2nd International Conference on Information Systems Security and Privacy* (2016), pp. 407–414. DOI: 10.5220/0005740704070414.

[9] Argha Ghosh and A Senthilrajan. "Research on Packet Inspection Techniques". In: *International Journal Of Scientific & Technology Research* 8 (2019), pp. 2068–2073.

[10] Auwal Sani Iliyasu and Huifang Deng. "Semi-Supervised Encrypted Traffic Classification With Deep Convolutional Generative Adversarial Networks". In: *IEEE Access* 8 (2020), pp. 118–126. DOI: 10.1109/ACCESS.2019.2962106.

[11] Manuel Lopez-Martin et al. "Network traffic classifier with convolutional and recurrent neural networks for Internet of Things". In: *IEEE Access* 5 (2017), pp. 18042–18050.

[12] Mohammad Lotfollahi et al. "Deep packet: a novel approach for encrypted traffic classification using deep learning". In: *Soft Computing* 24.3 (Feb. 2020), pp. 1999–2012. ISSN: 1433-7479. DOI: 10.1007/s00500-019-04030-2. URL: https://doi.org/10.1007/s00500-019-04030-2.

[13] Andreas Maurer, Massimiliano Pontil, and Bernardino Romera-Paredes. "The benefit of multitask representation learning". In: *Journal of Machine Learning Research* 17.81 (2016), pp. 1–32.

[14] Quinn McNemar. "Note on the sampling error of the difference between correlated proportions or percentages". In: *Psychometrika* 12.2 (1947), pp. 153–157.

[15] Shane Miller, Kevin Curran, and Tom Lunney. "Multilayer Perceptron Neural Network for Detection of Encrypted VPN Network Traffic". In: *2018 International Conference On Cyber Situational Awareness, Data Analytics And Assessment (Cyber SA)*. 2018, pp. 1–8. DOI: 10.1109/CyberSA.2018.8551395.

[16] Andrew W Moore and Konstantina Papagiannaki. "Toward the accurate identification of network applications". In: *International Workshop on Passive and Active Network Measurement*. Springer. 2005, pp. 41–54.

[17] Shahbaz Rezaei and Xin Liu. "Deep Learning for Encrypted Traffic Classification: An Overview". In: *IEEE Commun. Mag.* 57.5 (2019), pp. 76–81. DOI: 10.1109/MCOM.2019.1800819. URL: https://doi.org/10.1109/MCOM.2019.1800819.

[18] Shahbaz Rezaei and Xin Liu. "Multitask learning for network traffic classification". In: *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. IEEE. 2020, pp. 1–9.

[19] Hamid Tahaei et al. "The rise of traffic classification in IoT networks: A survey". In: *Journal of Network and Computer Applications* 154 (2020), p. 102538.

[20] Silvio Valenti et al. "Reviewing traffic classification". In: *Data Traffic Monitoring and Analysis*. Springer, 2013, pp. 123–147.

[21] Petr Velan et al. "A survey of methods for encrypted traffic classification and analysis". In: *International Journal of Network Management* 25.5 (2015), pp. 355–374. DOI: https://doi.org/10.1002/nem.1901. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nem.1901. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nem.1901.

[22] Radu Velea et al. "Network traffic anomaly detection using shallow packet inspection and parallel k-means data clustering". In: *Studies in Informatics and Control* 26.4 (2017), pp. 387–396.

[23] Wei Wang et al. "End-to-end encrypted traffic classification with one-dimensional convolution neural networks". In: *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*. 2017, pp. 43–48. DOI: 10.1109/ISI.2017.8004872.

[24] Jingjing Zhao et al. "Network traffic classification for data fusion: A survey". In: *Information Fusion* 72 (2021), pp. 22–47.

APPENDIX A
TEST RESULTS

TABLE 5
Contingency table for traffic type of the two multi-task classifiers.

CNN \ RNN	Correct	Incorrect
Correct	1506	55
Incorrect	46	30

TABLE 6
Contingency table for application type of the two multi-task classifiers.

CNN \ RNN	Correct	Incorrect
Correct	1511	62
Incorrect	28	36

TABLE 7
Contingency table for traffic type of the CNN multi-task and single-task classifiers.

CNN Multi \ CNN Single	Correct	Incorrect
Correct	1539	22
Incorrect	21	55

TABLE 8
Contingency table for application type of the CNN multi-task and single-task classifiers.

CNN Multi \ CNN Single	Correct	Incorrect
Correct	1536	37
Incorrect	19	45

APPENDIX B
LABEL DISTRIBUTION

TABLE 9
Label proportions. The last column contains "marginal" proportions of application type, the body of the table contains application proportions within traffic types.

Application	**Traffic type**							
	Chat	Email	File Transfer	P2P	Streaming	Video	VoIP	Proportions
AIM	0.010	0	0	0	0	0	0	0.001
Bittorrent	0	0	0	1.000	0	0	0	0.002
Facebook	0.030	0	0	0	0	0.260	0.045	0.050
FTPS	0	0	0.030	0	0	0	0	0.003
FTPS_down	0	0	0.358	0	0	0	0	0.035
Hangouts	0.177	0	0	0	0	0.435	0.361	0.270
ICQ	0.154	0	0	0	0	0	0	0.018
Mail	0	1.000	0	0	0	0	0	0.031
Netflix	0	0	0	0	0.353	0	0	0.028
SFTP	0	0	0.039	0	0	0	0	0.004
SFTP_down	0	0	0.006	0	0	0	0	0.001
SFTP_up	0	0	0.003	0	0	0	0	0.0003
Skype	0.629	0	0.565	0	0	0.305	0.219	0.282
Vimeo	0	0	0	0	0.644	0	0	0.050
Voipbuster	0	0	0	0	0	0	0.375	0.226
Youtube	0	0	0	0	0.004	0	0	0.0003