Network analytics project:
Implementation of “Deep Packet: a novel approach for encrypted traffic classification using deep learning” by Lotfollahi et al., 2019.

Benjamin Jones

1. Introduction

This document is a research and implementation project of a novel solution to a network analysis problem. The intention of this project is to show my ability to research new concepts, understand them, understand their uses, and implement them while achieving the same or similar results. All of the information about the experiment and the model of focus, Deep Packet, is not created by me and all credit goes to Mohammed Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsageh Saberian, the authors of the white paper that this project is based on (Lotfollahi et al., 2019).

2. Background

In the Deep Packet article, the authors aim to provide a solution to a network analysis problem using neural networks. The problem is that, as more internet traffic is encrypted, it becomes increasingly difficult to analyse networks for malicious traffic short of outright decrypting all traffic, which introduces privacy, ethical, and even legal concerns in many countries (Velan et al., 2015) (Bagui et al., 2017). The authors propose a complex Neural Network model consisting of a number of data pre-processing steps to turn packets from a pcap file into a long set of individual bytes, a Structured Auto-Encoder (SAE), and a Convolutional Neural Network (CNN) to which they call ‘Deep Packet’. I will be following their work and implementing my interpretation of how they have coded their model and, where they describe, use their exact specifications.

3. Methodology

3.1 Dataset

The dataset being used is the ISCX “VPN-nonVPN traffic” dataset from the University of New Brunswick (VPN-nonVPN dataset (ISCXVPN2016) 2021). Specifically the batches used from that collection are the “NonVPN-PCAPs” part 1 and the “VPN-PCAPS” parts 1 and 2. It is likely, although never explicitly stated in the original article, that the authors used the “samplePCAPs” batch based on the year both the dataset and the article were produced, but that file was corrupted when downloaded and used as of May 2023 and no alternative download is provided or hosted. Also corrupted were parts 2 and 3 of the “NonVPN-PCAPs” set, hence there are some differences in the amount and spread of the data categories used within this report compared to the original authors, but measures have been taken to ensure at least it is handled in the same way regarding pre-processing and defining classifications.

3.2 Pre-processing

The raw dataset contains a lot of irrelevant packets and data that needs to be stripped out before use within the Deep Packet model. The pre-processing and a reasoning for each step is as follows:

1. Remove Data-Link headers from all packets

   Reasoning: This header contains no consistent defining feature and will not contribute to the model’s accuracy in any positive manner.

2. Append “0” onto UDP headers until they meet the same size as TCP headers (20 bytes)

Reasoning: This keeps the headers of transport layer packets uniform in length.

3. Remove irrelevant packet types (DNS, TCP handshake, etc.)

   Reasoning: These packets do not contain the data looking to be classified.

4. Group all packets into byte-size chunks (1 byte per column)

   Reasoning: This cuts down the size of the data being fed into the model drastically.

5. Truncate all packets to the IP header and the next 1480 bytes, any packet payloads lower than 1480 bytes get "0" appended to the end until 1480 bytes is met

   Reasoning: The Maximum Transmission Unit size of most networks is 1500 bytes so all smaller packets should be brought up to a uniform size, rather than cutting off valuable data by bringing larger packets down to a uniform size.

6. All bytes are divided by 255

   Reasoning: By dividing each byte by the maximum value a byte can be, the byte will be condensed into a 0 or a 1 which will then represent that segment of the packet. Allowing a consistent, concise data feature for the model.

7. Mask the IP address in the IP header

   Reasoning: This avoids the model learning classifying traffic types based on the limited sample of hosts available within this dataset

Application	Size (K)
AIM chat	2
Email	19
Facebook	2475
FTPS	1
Hangouts	818
ICQ	1
Netflix	1
SFTP	2
Skype	758
Spotify	1
Torrent	6
Voipbuster	719
Vimeo	1
Youtube	2
Traffic Type	**Size (K)**
Chat	5
Email	19
Streaming	735
VoIP	1736
VPN: Chat	1
VPN: Email	1
VPN: File transfer	18
VPN: Torrent	6
VPN: Streaming	2
VPN: VoIP	2278

Table 1 Number of samples per Application (top) and traffic type (bottom) with classifier

The pre-process functions as described above are accomplished using a Python 3 script which performs each step to each application's packet capture file and outputs a .csv file of 1500 columns containing a 0 or a 1 per sample. The code for this script can be found in the Appendix (App. 1) or at (https://github.com/Bnjon01/DeepPacketNN/blob/main/DeepPacket_Process.py).

Following this, the files of sanitised data will then be grouped based on the application it was created by in one set of data and by traffic type in another. For each group, a value of 0 to 13 (for application classification) or 0 to 9 (for traffic classification) is appended to the final column of 1501 and this represents the classification value used for training data and for confirming predictions. This number is encoded into an array with one value being "1" and all others "0", of which the index of the former value is the original classifier value. Data is then randomly removed in categories with larger sample sizes until all categories are of relatively equal size, then finally, this data is split into three randomly selected groups of training data (64%), validation data (16%), and testing data (20%).

3.3 Model

After pre-processing is performed to completion, the next stage of Deep Packet is to feed the data into two different Neural Network architectures. One architecture is a Structured Auto-Encoder (SAE) and the other is a Convolutional Neural Network (CNN). An illustration by the authors can be seen in Figure 1, showing each of the two major portions of the model. Rather than one big program, I have opted for two separate scripts; one that handles the data pre-processing and another that contains the Neural Networks.

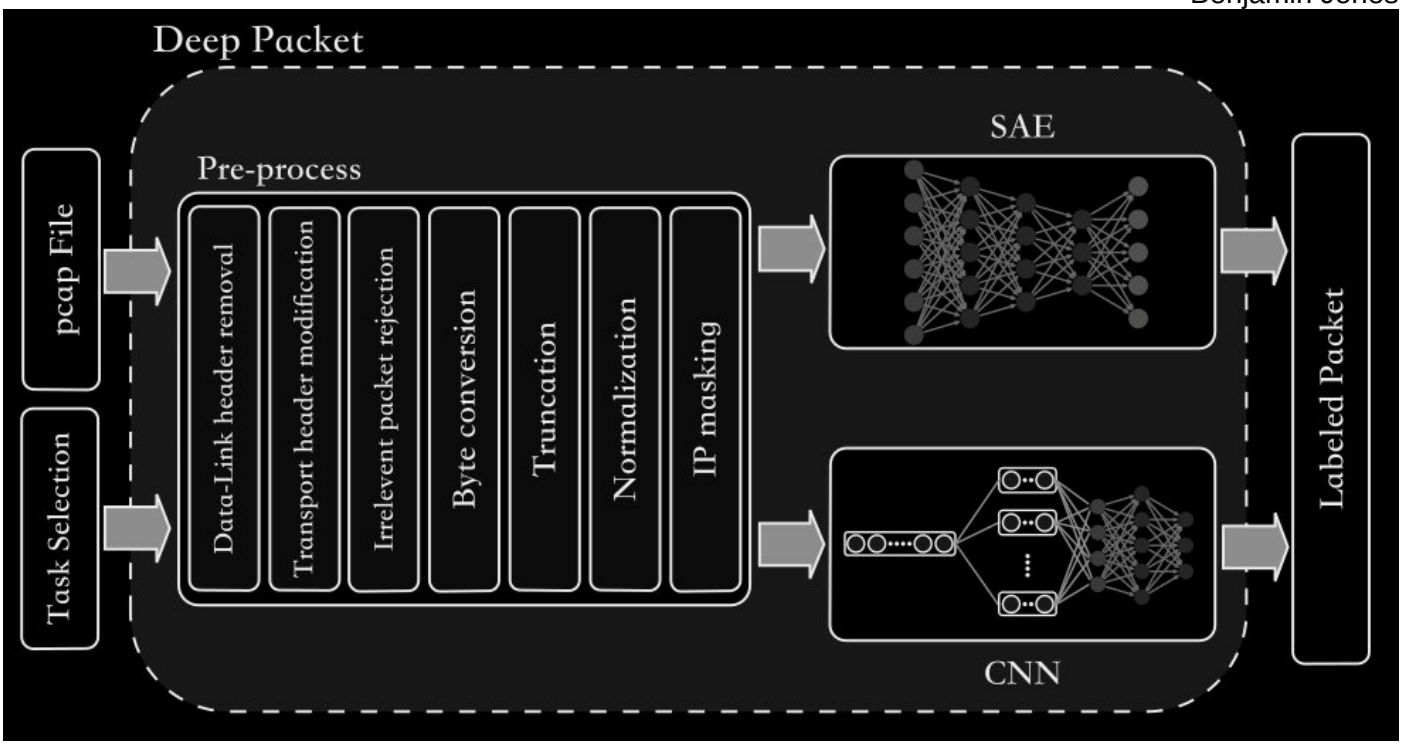

Figure 1 Deep Packet illustration (Lotfollahi et al., 2019)

For Deep Packet, the dataset is fed into both of the aforementioned network architectures individually and each comes to its own prediction results from the same sets of inputs, with both sets of results being the final output that is assessed. Like the original article, I have used the Keras interface with the TensorFlow back-end for the actual network implementation of both architectures.

The Stacked Auto-Encoder (SAE), is a complex architecture of several hidden layers to which each layer can be tweaked finely as it reduces the output options much more finely than a regular autoencoder (Vincent et al., 2008)

The SAE in Deep Packet contains five fully-connected layers containing 400, 300, 200, 100, and 50 neurons respectively and a dropout of 0.05 between each layer. The final output layer is a softmax classifier that contains either 10 or 14 neurons based on which classification is being trained and predicted. The SAE is first trained in a greedy layer-wise fashion with “Adam” as the optimiser and “mean square error” as the loss function over 200 epochs. It is then trained for a further 200 epochs but with the loss function changed to “categorical cross entropy”.

A 1-Dimensional Convolutional Neural Network (1D-CNN) a Neural Network architecture that has excelled in biology, machine visual, and facial recognition classifications (Lee et al., 2009) (Alipanahi et al., 2015) that generally includes, depending on the dimension aimed for, convolutional layers which are flattened in a pooling layer and fed into a more typical network of hidden layers (IBM, What are convolutional neural networks?).

Deep Packet’s CNN contains two consecutive convolutional layers, then a max pooling layer that feeds into a neural network of three fully-connected hidden layers, each layer also utilising the same dropout as the SAE. Finally the output, also like SAE, is a softmax classifier. The hyper parameters for the convolutional layers vary based on which classification is being performed, with the first being a kernel size of 4, number of layers at 200, and stride of 3 for application identification and a size of 5, number of 200, and a stride of 3 for traffic types. The second layer has a size of 5, number of 200, and stride of 1 for application identification and a size of 4, number of 200, and stride of 3 for traffic types. The CNN is trained over 300 epochs using using “categorical cross entropy” for a loss function and “Adam” for an optimiser.

For both architectures, all network hidden layers use “ReLU” for the activation function.

The code for this script can be found in the Appendix (App. 2) or at (https://github.com/Bnjon01/DeepPacketNN/blob/main/DeepPacket_NN.py).

3.4 Aim

The aim of this model is to be able to predict the application origin and traffic type of encrypted traffic within a network. Each architecture of the Deep Packet model individually outputs three values for assessment:

1. Recall – The percentage of True Positives found compared to True Positive plus False Negatives
2. Precision – The percentage of True Positives compared to all Positives
3. Weighted average F1 – A percentage value used to find the most optimal model hyperparameters that the Deep Packet authors could achieve, calculated as F1 = $\frac{2 * Recall * Precision}{Recall + Precision}$.

The goal of this project's outcome is to achieve comparable results to the original authors, which will show that my implementation is accurate enough for their description of the model they posit.

4. Results

Application Classification (SAE)			
	Precision	**Recall**	**F1-score**
AIM chat	0.96	0.98	0.97
Email	0.98	0.96	0.97
Facebook	1.00	1.00	1.00
FTPS	0.94	0.78	0.85
Hangouts	0.99	0.99	0.99
ICQ	1.00	0.81	0.90
Netflix	0.85	0.70	0.76
SFTP	0.59	0.83	0.69
Skype	1.00	1.00	1.00
Spotify	0.73	0.90	0.80
Torrent	1.00	1.00	1.00
Voipbuster	1.00	1.00	1.00
Vimeo	0.89	0.79	0.84
Youtube	0.91	0.90	0.90
Wtd. Average	**0.95**	**0.95**	**0.95**

Traffic Classification (SAE)			
	Precision	**Recall**	**F1-score**
Chat	0.95	0.99	0.97
Email	0.98	0.95	0.97
Streaming	0.97	1.00	0.99
VoIP	1.00	0.99	0.99
VPN: Chat	0.97	0.96	0.97
VPN: Email	1.00	0.98	0.99
VPN: File transfer	0.98	0.98	0.98
VPN: Torrent	1.00	0.99	0.99
VPN: Streaming	0.98	0.99	0.99
VPN: VoIP	1.00	1.00	1.00
Wtd. Average	**0.98**	**0.98**	**0.98**

Application Classification (CNN)			
	Precision	**Recall**	**F1-score**
AIM chat	0.96	0.96	0.96
Email	0.96	0.96	0.96
Facebook	1.00	1.00	1.00
FTPS	0.97	0.97	0.97
Hangouts	1.00	1.00	1.00

ICQ	0.94	1.00	0.97
Netflix	0.94	0.80	0.86
SFTP	1.00	1.00	1.00
Skype	1.00	0.99	1.00
Spotify	0.70	0.87	0.77
Torrent	0.99	0.99	0.99
Voipbuster	1.00	1.00	1.00
Vimeo	0.79	0.76	0.77
Youtube	0.93	0.89	0.91
Wtd. Average	**0.96**	**0.96**	**0.96**
Traffic Classification (CNN)			
	Precision	**Recall**	**F1-score**
Chat	0.95	0.94	0.95
Email	0.95	0.96	0.95
Streaming	0.98	1.00	0.99
VoIP	1.00	0.98	0.99
VPN: Chat	0.96	0.98	0.97
VPN: Email	1.00	1.00	1.00
VPN: File transfer	0.99	0.98	0.99
VPN: Torrent	1.00	1.00	1.00
VPN: Streaming	1.00	1.00	1.00
VPN: VoIP	1.00	1.00	1.00
Wtd. Average	**0.98**	**0.98**	**0.98**

Table 2 Results of Deep Packet in Traffic and Application classification

The results of both architectures within my implementation, shown in Table 2, is extremely close to the results obtained by the original authors which is a good sign that the data processing and Deep Packet implementation are at least close enough to the actual design made by the authors. The screenshots of the full SciKit Classification Reports can be found in the Appendix (App.3.1 App.3.2 App.3.3 App.3.4). The weighted average for Precision, Recall, and F1 are all within 0.05% (additively) which is extremely close to the original results obtained by the model's creators shown in the excerpts of the article, Figure 2 and Figure 3. Along with some variance of data input, randomness, and any minor architectural differences in the neural networks, this shows that my model is likely within about 94.8% similarity to the original authors based on weighted results. This similarity along with simply a good result percentage shows that this model, my understanding of it, and my implementation of it are a fairly reasonable success in the function of encrypted application and traffic classifications.

Table 3 Deep Packet performance for the application identification task

Application	CNN			SAE		
	Rc	Pr	F_1	Rc	Pr	F_1
AIM chat	0.76	0.87	0.81	0.64	0.76	0.70
Email	0.82	0.97	0.89	0.99	0.94	0.97
Facebook	0.95	0.96	0.96	0.95	0.94	0.95
FTPS	1.00	1.00	1.00	0.77	0.97	0.86
Gmail	0.95	0.97	0.96	0.94	0.93	0.94
Hangouts	0.98	0.96	0.97	0.99	0.94	0.97
ICQ	0.80	0.72	0.76	0.69	0.69	0.69
Netflix	1.00	1.00	1.00	1.00	0.98	0.99
SCP	0.99	0.97	0.98	1.00	1.00	1.00
SFTP	1.00	1.00	1.00	0.96	0.70	0.81
Skype	0.99	0.94	0.97	0.93	0.95	0.94
Spotify	0.98	0.98	0.98	0.98	0.98	0.98
Torrent	1.00	1.00	1.00	0.99	0.99	0.99
Tor	1.00	1.00	1.00	1.00	1.00	1.00
VoipBuster	1.00	0.99	0.99	0.99	0.99	0.99
Vimeo	0.99	0.99	0.99	0.98	0.99	0.98
YouTube	0.99	0.99	0.99	0.98	0.99	0.99
Wtd. average	**0.98**	**0.98**	**0.98**	**0.96**	**0.95**	**0.95**

Figure 2 Excerpt - Table 3 by (Lotfollahi et al., 2019)

Table 4 Deep Packet performance for the traffic characterization task

Class name	CNN			SAE		
	Rc	Pr	F_1	Rc	Pr	F_1
Chat	0.71	0.84	0.77	0.68	0.82	0.74
Email	0.87	0.96	0.91	0.93	0.97	0.95
File transfer	1.00	0.98	0.99	0.99	0.98	0.99
Streaming	0.87	0.92	0.90	0.84	0.82	0.83
Torrent	1.00	1.00	1.00	0.99	0.97	0.98
VoIP	0.88	0.63	0.74	0.90	0.64	0.75
VPN: Chat	0.98	0.98	0.98	0.94	0.95	0.94
VPN: File transfer	0.99	0.99	0.99	0.95	0.98	0.97
VPN: Email	0.98	0.99	0.99	0.93	0.97	0.95
VPN: Streaming	1.00	1.00	1.00	0.99	0.99	0.99
VPN: Torrent	1.00	1.00	1.00	0.97	0.99	0.98
VPN: VoIP	1.00	0.99	1.00	1.00	0.99	0.99
Wtd. average	**0.94**	**0.93**	**0.93**	**0.92**	**0.92**	**0.92**

Figure 3 Excerpt - Table 4 by (Lotfollahi et al., 2019)

5. Conclusion

In this report, I've shown my understanding of new concepts, my ability to research them, and my ability to re-create them. This report and the white paper it is based off discusses an extremely important topic in network forensics and analytics and one that is increasing in importance year over year. This Deep Packet model is shown to be extremely effective at classifying a wide variety of both traffic types as well as specific applications they are created from all without compromising the privacy and security that end-to-end encryption allows.

This model or models that achieve similar results could be used to detect attack vectors or data flow issues with much more clarity within a network than simple blind maneuvers, bringing a level of precision to correcting issues such as attacks, malfunction, or simple bug fixing, all the while still following data privacy laws such as the Privacy Act (1988) or the GDPR.

References

Lotfollahi, M. *et al.* (2019) ‘Deep packet: A novel approach for encrypted traffic classification using Deep Learning’, *Soft Computing*, 24(3), pp. 1999–2012. doi:10.1007/s00500-019-04030-2.

Velan, P. *et al.* (2015) ‘A survey of methods for encrypted traffic classification and analysis’, *International Journal of Network Management*, 25(5), pp. 355–374. doi:10.1002/nem.1901.

Bagui, S. *et al.* (2017) ‘Comparison of machine-learning algorithms for classification of VPN network traffic flow using time-related features’, *Journal of Cyber Security Technology*, 1(2), pp. 108–126. doi:10.1080/23742917.2017.1321891.

VPN-nonVPN dataset (ISCXVPN2016) (2021) *University of New Brunswick est.1785*. Available at: https://www.unb.ca/cic/datasets/vpn.html (Accessed: 26 May 2023).

Vincent, P. *et al.* (2008) ‘Extracting and composing robust features with denoising autoencoders’, *Proceedings of the 25th international conference on Machine learning - ICML ’08* [Preprint]. doi:10.1145/1390156.1390294.

Lee, H. *et al.* (2009) ‘Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations’, *Proceedings of the 26th Annual International Conference on Machine Learning* [Preprint]. doi:10.1145/1553374.1553453.

Alipanahi, B. *et al.* (2015) ‘Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning’, *Nature Biotechnology*, 33(8), pp. 831–838. doi:10.1038/nbt.3300.

What are convolutional neural networks? (no date) *IBM*. Available at: https://www.ibm.com/topics/convolutional-neural-networks (Accessed: 26 May 2023).

Appendix

App. 1 Data pre-processing script

```python
from scapy.all import *
import binascii
import csv

# Returns the packet at the IP header, removing everything above/prior to it
def data_link_removal(packet):
        return packet[IP]

# Converts the packet into its Hexcode form and returns an array with each byte in an index
def byte_conversion(packet):
        array = []
        hex_var = str(binascii.hexlify(bytes(packet)))[2:-1]
        for i in range(len(hex_var))[::2]:
                array.append(hex_var[i:i+2])
        return array

# Checks if the packet is UDP and if so, appends 12 bytes of 00 to match the 20 byte header of TCP
def transport_modification(array):
        if UDP in array:
                array = array[:28] + ["00", "00", "00", "00", "00", "00", "00", "00", "00", "00", "00", "00"] + array[28:]
        return array

# Divides each byte in an array by 255 to make it either a 0 or 1 value
def normalisation(array):
        for i in range(len(array)):
                array[i] = round(int(output[i], 16)/255)
        return array

# Returns the given array with a full length of 1500 bytes, if it is not large enough, appends 0 until 1500 is met and then returns that
def truncation(array):
        while len(array) < 1500:
                array.append(0)
        return array[:1500]

# changes the source and destination ip addresses in the ip header to 0
def ip_masking(array):
        for i in range(25, 33):
                array[i] = 0
        return array

if __name__ == "__main__":
        # input pcap or pcapng file location
        filename = "filename.pcap"
        # Output name of the processed file ending in .csv
        output_filename = "filename.csv"
        # The number of category of file to be appended to the final column
        category_num = 0
        pcap = rdpcap(filename)
        with open(output_filename, 'w') as csvfile:
                csvwriter = csv.writer(csvfile)
                for packet in pcap:
                        # Skips the loop if DNS, ARP, or other non-IP packet is found
                        if DNS in packet or ARP in packet or IP not in packet:
                                continue
                        # Skips the loop if TCP packet contains either SYN, ACK, or FIN flags
                        elif TCP in packet:
                                if "S" in packet[TCP].flags or "A" in packet[TCP].flags or "F" in packet[TCP].flags:
                                        continue
                        output = data_link_removal(packet)
                        output = byte_conversion(output)
                        output = transport_modification(output)
                        output = normalisation(output)
                        output = truncation(output)
                        output = ip_masking(output)
                        output.append(category_num)
                        csvwriter.writerow(output)
```

App. 2 Deep Packet Neural Network code

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, MaxPooling1D, Conv1D, Flatten
from tensorflow.keras import metrics
from keras.utils import np_utils
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

# Data file input, comment/uncomment the given csv file or replace the name
#data = pd.read_csv("TrafficData.csv", header=None, delimiter=',', skiprows=2)
data = pd.read_csv("ApplicationData.csv", header=None, delimiter=',', skiprows=2)

# Split data into X and Y, cutting off column 1501 for Y
X_data = data.iloc[:,:1500]
y_data = data.iloc[:,1500:]
# Encode Y into an array, 10 for traffic classification, 14 for application classification
y_data = np_utils.to_categorical(y_data, 14)

# 64% training data, 20% test data, 16% validation data
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size = 0.2)
X_train, X_val, y_train, y_val = train_test_split(X_data, y_data, test_size = 0.2)

#SAE
SAE = Sequential()
SAE.add(Dense(400, input_shape=(1500,), activation="relu"))
SAE.add(Dropout(0.05))
SAE.add(Dense(300, activation="relu"))
SAE.add(Dropout(0.05))
SAE.add(Dense(200, activation="relu"))
SAE.add(Dropout(0.05))
SAE.add(Dense(100, activation="relu"))
SAE.add(Dropout(0.05))
SAE.add(Dense(50, activation="relu"))
SAE.add(Dropout(0.05))
SAE.add(Dense(14, activation="softmax"))
# Training with MSE loss function
SAE.compile(loss="mean_squared_error", optimizer="adam")
SAE.fit(X_train, y_train, epochs=200, verbose=1)
# Training with CCE loss function
SAE.compile(loss="categorical_crossentropy", optimizer="adam", metrics=[metrics.Precision(), metrics.Recall()])
SAE.fit(X_train, y_train, epochs=200, verbose=1)

y_pred = np.argmax(SAE.predict(X_test), axis=-1)
y_pred = np_utils.to_categorical(y_pred, 14)
print(classification_report(y_test, y_pred))

#CNN
CNN = Sequential()
#Application classification layers
CNN.add(Conv1D(200, input_shape=(1500,1), activation="relu", kernel_size=4, strides=3))
CNN.add(Conv1D(200, activation="relu", kernel_size=5, strides=1))

##Traffic classification
#CNN.add(Conv1D(200, input_shape=(1500,1), activation="relu", kernel_size=5, strides=3))
#CNN.add(Conv1D(200, activation="relu", kernel_size=4, strides=3))

CNN.add(MaxPooling1D())
CNN.add(Flatten())
CNN.add(Dense(200, activation="relu"))
CNN.add(Dropout(0.05))
CNN.add(Dense(200, activation="relu"))
CNN.add(Dropout(0.05))
CNN.add(Dense(200, activation="relu"))
CNN.add(Dropout(0.05))
CNN.add(Dense(14, activation="softmax"))
CNN.compile(loss="categorical_crossentropy", optimizer="adam", metrics=[metrics.Precision(), metrics.Recall()])
CNN.fit(X_train, y_train, epochs=300, verbose=1)

y_pred = np.argmax(CNN.predict(X_test), axis=-1)
y_pred = np_utils.to_categorical(y_pred, 14)
print(classification_report(y_test, y_pred))
```

App. 3 Neural Network Results

App. 3.1 CNN result for Application Analysis

```
60/60 [==============================] - 1s 19ms/step
              precision    recall  f1-score   support

           0       0.96      0.96      0.96       203
           1       0.96      0.96      0.96       213
           2       1.00      1.00      1.00       191
           3       0.97      0.97      0.97        59
           4       1.00      1.00      1.00       196
           5       0.94      1.00      0.97        15
           6       0.94      0.80      0.86        60
           7       1.00      1.00      1.00        12
           8       1.00      0.99      1.00       211
           9       0.70      0.87      0.77        91
          10       0.99      0.99      0.99       195
          11       1.00      1.00      1.00       188
          12       0.79      0.76      0.77        41
          13       0.93      0.89      0.91       215

   micro avg       0.96      0.96      0.96      1890
   macro avg       0.94      0.94      0.94      1890
weighted avg       0.96      0.96      0.96      1890
 samples avg       0.96      0.96      0.96      1890
```

App. 3.2 CNN result for Traffic Analysis

```
55/55 [==============================] - 0s 7ms/step
              precision    recall  f1-score   support

           0       0.95      0.94      0.95       193
           1       0.95      0.96      0.95       221
           2       0.98      1.00      0.99       185
           3       1.00      0.98      0.99       187
           4       0.96      0.98      0.97        98
           5       1.00      1.00      1.00        47
           6       0.99      0.98      0.99       199
           7       1.00      1.00      1.00       207
           8       1.00      1.00      1.00       209
           9       1.00      1.00      1.00       209

   micro avg       0.98      0.98      0.98      1755
   macro avg       0.98      0.98      0.98      1755
weighted avg       0.98      0.98      0.98      1755
 samples avg       0.98      0.98      0.98      1755
```

App. 3.3 SAE result for Application Analysis

```
60/60 [==============================] - 0s 816us/step
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       199
           1       0.98      0.96      0.97       194
           2       1.00      1.00      1.00       196
           3       0.94      0.78      0.85        40
           4       0.99      0.99      0.99       197
           5       1.00      0.81      0.90        16
           6       0.85      0.70      0.76        56
           7       0.59      0.83      0.69        12
           8       1.00      1.00      1.00       203
           9       0.73      0.90      0.80       110
          10       1.00      1.00      1.00       219
          11       1.00      1.00      1.00       190
          12       0.89      0.79      0.84        62
          13       0.91      0.90      0.90       196

   micro avg       0.95      0.95      0.95      1890
   macro avg       0.92      0.90      0.91      1890
weighted avg       0.96      0.95      0.95      1890
 samples avg       0.95      0.95      0.95      1890
```

App. 3.4 SAE result for Traffic Analysis

```
55/55 [==============================] - 0s 797us/step
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       210
           1       0.98      0.95      0.97       202
           2       0.97      1.00      0.99       179
           3       1.00      0.99      0.99       211
           4       0.97      0.96      0.97       112
           5       1.00      0.98      0.99        41
           6       0.98      0.98      0.98       192
           7       1.00      0.99      0.99       212
           8       0.98      0.99      0.99       194
           9       1.00      1.00      1.00       202

   micro avg       0.98      0.98      0.98      1755
   macro avg       0.98      0.98      0.98      1755
weighted avg       0.98      0.98      0.98      1755
 samples avg       0.98      0.98      0.98      1755
```

App. 4 Processed data size