Department of Computer Science

Submitted in part fulfilment for the degree of MEng.

Abnormal trajectory detection with deep learning

Dragos Stoican

2023-May-3

Supervisor: Kofi Appiah

Contents

Part I

Executive Summary

1 Executive summary

The increasing amount of availability of large amounts of data has caused deep learning to gain prominence in numerous fields, augmenting and enhancing other machine-learning techniques in some applications. These capabilities have led to breakthroughs in computer vision, natural language processing, and among many others, anomaly detection. Anomaly detection consists of identifying rare or abnormal events or patterns and is crucial in many real-world applications, ranging from fraud detection to fault diagnosis, or as I explore in this project, abnormal trajectory detection. In recent years, deep learning has emerged as a promising approach for detecting abnormal trajectories in various fields like surveillance, traffic management, and healthcare, thanks to its ability to learn patterns and relationships from complex trajectory data and its various features.

The main objective of this project is to propose methods of detecting abnormal trajectories in traffic surveillance scenarios using deep neural networks. By creating multiple such architectures, my goal is to explore the current deep learning methodologies and their efficiency in abnormal trajectory detection. I aim to cover unsupervised, supervised, and semi-supervised approaches, various neural network architectures, data preprocessing techniques, and in-depth hyperparameter tuning. I intend to test the networks thoroughly using commonly used datasets in literature and draw conclusions on performance compared to state-of-the-art models. I aim to draw conclusions and justify methodologies based on training and testing results by presenting relevant statistical data. Ultimately, I aspire to conduct this research with the intention of facilitating real-world applications and taking into account implications such as cost efficiency and human intervention.

The approach taken follows the following steps:

- Research current approaches, identifying similarities in architectures
- Identify a suitable dataset to adopt and use for building and testing deep learning networks
- Reproduce a similar model to one of the identified ones to use as a baseline comparison for other novel implementations
- Create one or more novel architectures justified by the current liter-

ature, preferably using a completely different approach to the one reproduced in the previous objective

- Use the different architectures to paint a picture of the vast number of deep learning methodologies in abnormal trajectory detection
- Compare novel architectures to the reconstructed model, draw conclusions on performance, justify differences, tune training parameters
- Create custom datasets to further explore networks performance
- Highlight implications of real-world implementations of such systems and the cost efficiency of deployment
- Suggest further works and possible extensions of this research

Therefore, I suggest three deep-learning networks to explore the problem of abnormal trajectory detection:

- Reconstructed-DAE: A reconstruction of an unsupervised learning approach using a deep autoencoder explored in literature
- Two-Input-RNN: A novel Two Input Recurrent Neural Network, designed by myself as a supervised learning approach to the problem
- DBSCAN-RNN: A combination of the clustering algorithm DBSCAN and the Two-Input-RNN network created by myself, resulting in a semi-supervised learning network

The results obtained show great performance for the Two-Input-RNN novel architecture I designed, and a slightly worse performance with its semi-supervised counterpart, DBSCAN-RNN, but both performed better than the baseline comparison approach, the Reconstructed-DAE. However, the results don't cover only final accuracies, but they refer back to the objectives by discussing training metrics to explore the inner workings of each architecture. Due to the large range of techniques adopted across the three models, I am able to discuss and justify the usage of various loss functions, optimizers, data preprocessing, cross-validation, neural network layers, clustering algorithm parameters, and other fine-tuning aspects. Ultimately I create a custom dataset to solve the problem of abnormal trajectory detection in an intersection in York, and I use the results to showcase vulnerabilities of the networks and highlight real-world implications that go beyond the network performance.

This topic of study, with the approach I've taken shows no legal, social, ethical, or commercial issues as it only represents research material into the world of abnormal trajectory detection using deep learning

Part II

Main Body

1 Introduction

1.1 Motivation

Abnormal trajectory detection is a crucial task that can be utilized in various fields such as surveillance, to apprehend offenders during criminal acts, like leaving the house during house arrest [1] or trespassing in prohibited waters [2], [3]; transportation to improve autonomous systems [4] or traffic management [5], [6], and healthcare to detect early signs of disease [7]. The ability to identify abnormal patterns of movement can be a critical aspect in providing enhanced safety, efficiency, and well-being, and therefore it has become a popular research field, especially since the rise of deep learning techniques.

In addition, due to the exponentially increasing ability to collect user data, many end users struggle to find the most efficient way of interpreting it. This statement is no different when it comes to trajectory data. Over the last decade, the problem of trajectory classification, and in turn abnormal trajectory detection, has become more and more popular due to the amount of data now available, as suggested by J. Bian et. al. survey [8]. As traditional machine learning methods for anomaly detection like isolation forest [9] or clustering algorithms [10] fail to interpret large amounts of multi-dimensional data, research is moving towards more powerful, more computationally expensive deep learning models [5], [1], [11], [12]. These models can be tailored to the particular sequential structure of trajectory data and can take advantage of multiple spatio-temporal features to produce excellent results.

1.2 Objectives and report structure

In this project, I propose and analyze various methods for abnormal trajectory detection in traffic surveillance scenarios. The aims of this analysis will be as follows:

- Present the most viable state-of-the-art solutions

- Tackle the challenge of working with big data
- Tailor models to sequential data
- Demonstrate the potency and efficacy of deep learning methodologies
- Compare benefits of supervised, semi-supervised and unsupervised learning
- Discuss the intricacies of implementing such systems in real-world scenarios

The report will start by presenting a few of the most popular approaches for abnormal trajectory detection in the past 20 years via a literature review 1.2, focusing on similarities between these approaches and the reasoning of the authors. In the methodology chapter 2.3 I describe in detail the three methods of detection I've created for solving this challenge, and in the results section 3.4.2 I present the performance of the models and conduct a thorough comparison between the three models. Ideas for further development will be mentioned in the conclusion 4.6.

2 Literature review

2.1 Deep Autoencoders and Generative Adversarial Networks

In 2018, Roy and Bilodeau propose a solution [5] for detecting abnormal trajectories in road intersections, using a neural network model called a deep autoencoder (DAE) [13]. These fully connected networks can be used to learn the normality of the input data, and then a reconstruction threshold can be used to detect abnormalities when testing. The large amount of data used in the process is created by applying data augmentation techniques on a small dataset. This is due to the fact that abnormal data is very rare, and thus it has to be generated. The main steps taken by this detection model are shown in Figure 1.1. Roy and Bilodeau compare this approach to older machine learning techniques, like One-class SVM [14], Isolation forest [9], and vanilla autoencoders [13] and discover that it outperforms all in True Positive Rate and False Positive Rate for normal and abnormal

trajectories.

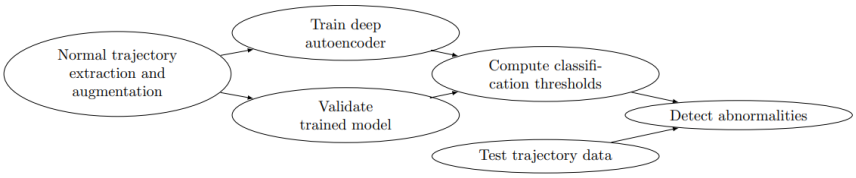

Figure 2.1: The main steps of abnormal trajectory detection using a DAE

One year later, Roy and Bilodeau [6] continue their research by improving their implementation in [5]. The DAE architecture is built upon by adding a discriminator inspired by Generative Adversarial Networks (GAN) [15]. When considering abnormal trajectory detection, normal GAN training techniques are unstable since the generator G has to generate realistic trajectories that are ordered sequences of points that must respect spatial and temporal constraints in order to fool the discriminator D, but in most cases, this is unachievable due the complexity of the data. Instead, as described in Figure 2.2, the generator generates a trajectory reconstruction error that reassembles the one obtained by the DAE in [5]. Results show that the Adversarially Learned Reconstruction Error Classifier (ALREC) outperforms their 2018 DAE network on any dataset, with the added benefit that it requires no manually defined reconstruction threshold.

An AE is also used by M. Ribeiro et al. in [11], but this time following a convolution network architecture to detect anomalies in videos, which can be seen in Figure 2.3. The approach is similar to that of Roy and Bilodeau's DAE [5], as the network aims to reconstruct video frames and use a reconstruction error as an anomaly detector. The difference here is that the dataset is a series of frames instead of spatial coordinates representing a trajectory. This works best for convolutional layers which have achieved state-of-the-art performance for object recognition in images, and it offers access to larger datasets, as this type of data is significantly easier to obtain. However, images don't benefit from the same sequential structure that trajectory data has, and, as proven by many others [1], this structure can be exploited by other deep learning methods.

Gupta et. al. also propose a GAN for predicting future behaviors of pedestrians in [4]. Their model differs from others by predicting multiple "good" trajectories that a pedestrian might take, as opposed to finding the most probable one. The novelty lies in the generator's capability to model human-human interactions, and in the custom loss function that encourages the production of multiple diverse trajectories. The experiments suggest that the Social GAN is especially good for long-term predictions, which is due to the addition of Long Short-Term Memory (LSTM) [16], [17] layers,

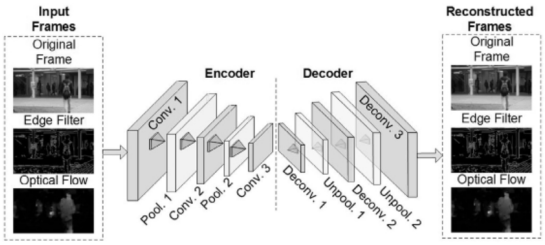

Figure 2.2: The aim of this network is to output reconstructed frames that are as similar as possible to the input frames. The hidden layers hold knowledge about how to interpret images, or how to recognize features in them

which specialize in making use of long sequences of data, as shown in other papers discussed here [1], [18].

2.2 Recurrent Neural Networks (RNN) with Long Short-Term Memory (LSTM) layers

Long Short-Term Memory (LSTM) networks, introduced by S. Hochreiter in [16], and further explored by K. Greff or Alex Sherstinsky in [17] and [19] respectively, have emerged as an effective and scalable model for several learning problems related to sequential data. These networks have been used to advance the state of the art in the fields of natural language processing, speech recognition, or translation, by solving the problem of exploding and vanishing gradients that RNNs usually faced. Due to these promising results, they have also become common occurrences in applications involving trajectory data, which share the same sequential nature.

In the work of N. Freitas [1] the challenges of working with trajectory data are clearly described. The four main challenges that can be applied to any trajectory classification problem are (1) the massive volume of data; (2) the complexity of the data; (3) the sparsity of the data; (4) the nature of multiple dimensions. Therefore, more complex models of neural networks are required to interpret data of this complexity and size. Freitas uses a state-of-the-art architecture (Figure 2.4) of RNNs, utilizing LSTMs to deal with these challenges. The model uses embedding layers to reduce the dimensionality of the data, which is then concatenated and passed into LSTM layers that learns complex patterns from sequences. The main takeaway from the results is that exploiting the sequential nature of the

data using RNNs that specialize in this structure, especially when multiple features are used, can turn the described challenges into unique benefits.

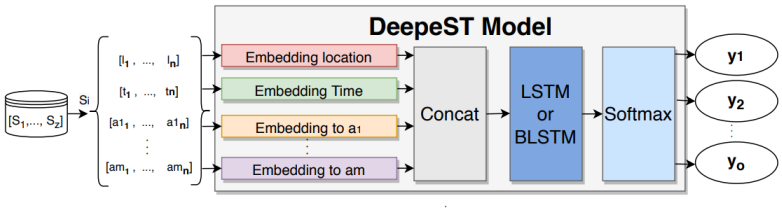

Figure 2.3: Subtrajectories S_i contain multiple sequential features, which are embedded and concatenated into a single array that is inputted into the LSTM layer, whose output can be used to classify the data

In [18] a similar LSTM prediction network called seq2seq is used to model normal trajectories. Yufan Ji also mentions here the excellent learning ability of LSTM networks in sequence data due to its structural characteristic, mentioning how recent research has shown this method can be a good choice for exploring and solving problems about movement. The innovative idea is to move away from the need of hand-crafted feature extraction, which recent studies depend more and more on as their models become more capable of making use of big data, like N Freitas also implies in [1]. Instead, they use deep learning algorithms to obtain sequence-type trajectory data via RNNs, removing the need to extract a large number of features manually. The spatio-temporal and semantic information of a generated trajectory can then be compared to input data to detect anomalies.

Tao Zhang [2] proposes an LSTM-based neural network combined with the clustering algorithm DBSCAN [10]. Unlike all the other papers discussed, this is a supervised learning approach, but since labeling such a large amount of data would be ineffective, the first step of the algorithm is clustering the trajectories based on speed and distance between each other, using DBSCAN. The trajectories that don't fit in any of the clusters are marked as abnormal trajectories, while the rest are labeled as normal trajectories. Then the problem then becomes a binary classification task for an LSTM network on the newly created dataset of normal and abnormal trajectories. The entire framework can be seen in Figure 2.5.

Perhaps the most relevant results of an abnormal trajectory detection LSTM network are the ones obtained by A Nanduri in [12]. The direct comparison he makes between various types of RNNs and the one-class SVM algorithm MKAD on a very specific dataset of aircraft trajectories show that these deep neural networks outperform machine learning approaches in all cases. Like Freitas in [1], he also demonstrates that older techniques suffer from the curse of dimensionality of the data, having to reduce the dimensions

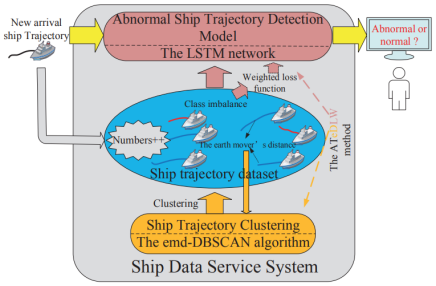

Figure 2.4: The DBSCAN algorithm clusters trajectories based on the earth mover's distance, and the velocities of boats. The clustered dataset is used to train the LSTM network, but due to class imbalance, a weighted loss function has to be used. New trajectories can then be classified by the trained model

through complex, time-consuming algorithms instead of making use of every feature.

2.3 Other methods

Another form of abnormal trajectory detection is presented by I. Kontopoulos et al. in [20]. This involves the use of computer vision techniques to classify trajectories into a set number of classes. Trajectory data would have to be represented in an image, and those images would then be used by a color histogram feature extractor, and a random forest classifier [21] to classify the state of the boat, and potentially identify abnormal behaviors.

3 Methodology

Based on the literature in the field, and the current state-of-the-art models, I propose three approaches for solving the problem of abnormal trajectory detection. The first method, called Reconstructed-DAE, will be a modified version of Roy and Bilodeau's deep autoencoder (DAE) network in [5], and it will serve as a benchmark for comparison of other methods. This unsupervised learning implementation will also be used in an attempt to justify Roy and Bilodeau's [5] choices of training hyperparameters like the

optimizer or loss function. The second method is a novel two-input binary classifier using Recurrent Neural Networks called Two-Input-RNN, which uses supervised learning to detect abnormal trajectories. The third method builds upon the novel Two-Input-RNN but it adds a DBSCAN classifier at the start of the workflow to create a semi-supervised learning approach that doesn't require a labeled dataset, I've called this one DBSCAN-RNN. The differences between these models will allow us to investigate the power of RNNs with sequential data, which is praised in [1], the limitations and benefits of unsupervised, semi-supervised, and supervised learning both in performance but also ease of deployment, and the best hyperparameter choices for either case.

3.1 Datasets

The datasets used are adopted from [5], and are composed of trajectory data collected in four road intersections with pedestrians, vehicles, and bicycles in the locations Sherbrooke, Rouen, St-Marc, and Rene-Levesque. The original data comes from the Urban Tracker Dataset [22], but due to the small number of trajectories and the lack of abnormal data, a lot of pre-processing is necessary before it can be used for the training and testing of deep neural networks.

The three networks I am presenting in this project make use of the already processed and augmented data that includes 10000 to 40000 entries for each intersection. However, let's explore the contents of the augmented data and the processes that lead to its creation. Figure 3.1 shows the original data included in the Urban Tracker Dataset [22], where each dataset consists of no more than 1000 entries of normal trajectories only, which is insufficient for training a deep neural network.

Pre-processing then consists of two steps, creating abnormal data and augmenting existing normal data. The former is done by creating two sets of abnormal entries:

- The first set consists of straight lines with constant velocities that could be considered "clearly abnormal". This is called "abnormal data".
- The second set contains more realistic abnormal trajectories inspired by real ones. These are created by applying transformations to the original dataset, maintaining consistent variability in both position and velocity while creating harder examples for networks to tackle. This is called "Real-Abnormal-Data". For each dataset, two sets of such

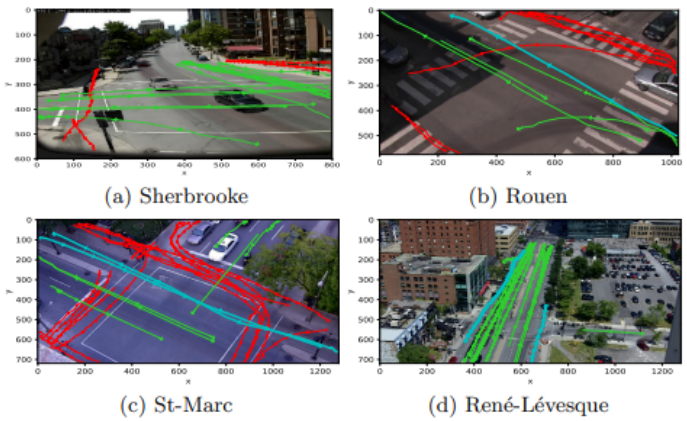

Figure 3.1: Original annotated trajectories of the Urban Tracker Dataset (image taken from [22]). Red, green, and blue represent pedestrians, cars, and bikes respectively. Note that Sherbrooke contains no bicycles and Rene contains no pedestrians. All trajectories are considered to be normal.

realistic abnormal data are created.

Figures 3.2 and 3.3 show the two abnormal subsets of trajectories for the St-Marc intersection. Abnormal trajectories for all the other locations can be seen in Appendix A. These figures were generated using the trajectory_viewer.py script, written by the author of this project and included in the code folder.

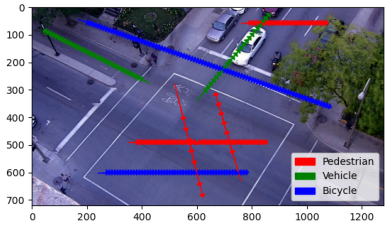

Figure 3.2: Simple abnormal data for the St-Marc dataset. Consists of jaywalking pedestrians crossing the road at inappropriate locations, cars on the cycle path or on the wrong side of the road, and bicycles cutting across the street or on the pavement

Training a deep neural network requires much more data than what is available in the Urban Tracker Dataset [22], which is why augmentation

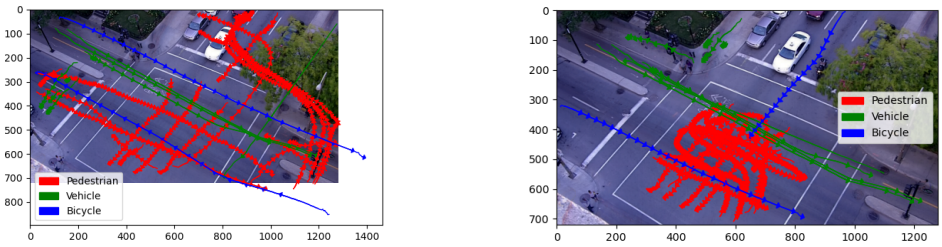

Figure 3.3: The realistic sets of abnormal trajectories, with pedestrians on the road, cars on the cycle path or on the pedestrian crossings, and bicycles on the pavement or on the road. These trajectories are more complex in velocities and are usually not only straight lines

techniques had to be used by Roy and Bilodeau [5] when they created the datasets. For every entry in the original data, 50 new trajectories are created by applying slight transformations to the spatial and velocity features. Furthermore, each trajectory is split into sub-trajectories composed of 31 timesteps of recorded position and velocity, as seen in Appendix B. This means that networks trained on these datasets don't require a full trajectory for classification, and could, in theory, be used with sub-trajectories generated in real-time by a surveillance system. Figure 3.4 shows the dimensions of the data at the end of the pre-processing stage. The plots were generated using the explore_dataset.py script, written by the author of this project and included in the code folder.

A trajectory within the dataset is represented by a vector of size 125 following the format [id, category, x1, y1, vx1, vy1, x2, y2, vx2, vy2, ..., x31, y31, vx31, vy31], where x and y are spatial coordinates and vx and vy are velocities at any of the 31 timesteps. The id represents which trajectory the sub-trajectory is part of, and the category is 0 for pedestrians, 1 for vehicles, and 2 for bicycles. Note that this representation of the data is not preserving the sequential nature of trajectory data, as consecutive elements in the array are not necessarily related. A more detailed view of this data format can be seen in Appendix C.

While the data is now in the correct shape and contains all the entries needed for training and testing, Roy and Bilodeau's [5] datasets had to be normalized before they can be used by any model. I used a min-max scaler to transform the data to values between 0 and 1. This is required because the trajectory data doesn't follow any specific probabilistic distribution, in fact, features in the trajectory data will have different ranges, like world coordinates and velocity, which are measured on different scales. Also,

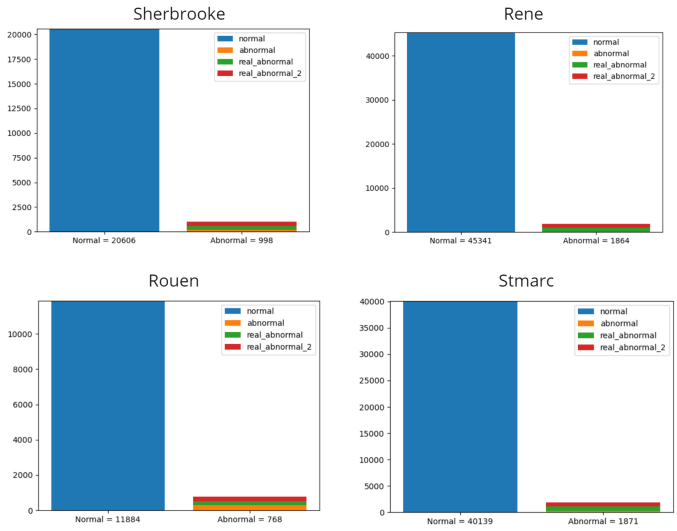

Figure 3.4: Data distribution between normal and abnormal trajectories for each dataset. Interesting to note the high class imbalance, but this is due to the fact that only normal data is used for training. Abnormal data is only necessary for testing and so it doesn't require the same size

this helps the network learn useful features faster and negates the effect of outliers while preserving the shape of the original distribution [23]. Appendix D includes experiments conducted with and without normalization and these show that without normalization the network is unable to learn, and the loss function stays the same during training. Data normalization was implemented in the data_preprocessing.py script, written by the author of this project and included in the code folder.

3.2 First approach (Reconstructed-DAE)

3.2.1 Autoencoders

An autoencoder is a type of fully-connected artificial neural network that is used for unsupervised learning [13], [24]. The goal of an autoencoder is to learn a compressed representation of input data, that captures the most important information while discarding redundant or irrelevant information. This is known as the Compressed Feature Vector (CFV) and it is produced by an encoding function represented by the encoder. The CFV is then passed through a series of layers that "decode" it back into a reconstruction of the original input data. A reconstruction loss is calculated based on

the difference between the input and the output, and the training consists of minimizing this loss. The simplest form of an autoencoder, consisting of only one hidden layer (the CFV) can be seen in Figure 3.5. A deep autoencoder [24] works like any other autoencoder, but it consists of a lot more layers, making it better suited for working with big data.

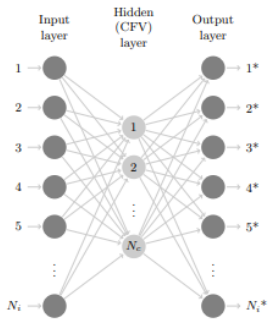

Figure 3.5: Diagram showing the simplest autoencoder.

3.2.2 Deep autoencoder for abnormal trajectory detection

The process of classifying abnormal trajectories using a deep autoencoder can be seen in Figure 3.6 and it is composed of the following four steps:

1. Train the DAE on the pre-processed normal trajectories by minimizing the error between the input and the output vectors. The neural network then specializes in reconstructing normal data.

2. Score the performance by calculating the mean error between the original trajectories and the reconstructed trajectories.

3. Calculate a reconstruction threshold, based on the model's performance in reconstructing normal data.

4. When testing, if the reconstruction error is higher than the threshold, the trajectory is abnormal, if the error is smaller than the threshold, the trajectory is normal.

All of these steps, and the result generation and analysis are done in the Reconstructed-DAE.ipynb jupyter notebook written by the author of this project and included in the code folder. For the creation of this model, the Pytorch [25] framework was used.

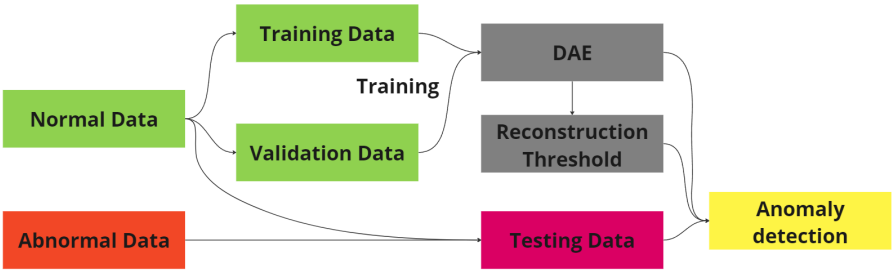

Figure 3.6: The workflow for abnormal trajectory detection using a deep autoencoder. Normal data is split into training and validation sets to train the model and compute a reconstruction threshold based on its performance. Both normal and abnormal data are then used for testing as any trajectory that produces an error larger than the threshold is labeled abnormal.

3.2.3 Training and computing the threshold

For training, only normal data is used, but to avoid over-fitting, I applied a k-fold cross-validation technique [26]. This means splitting the training data into k folds of approximately equal size. Each fold is chosen in turn for validation, while the remaining are used for training. In theory, a higher number of folds will produce a lower validation error for the model, however, this comes at the cost of execution time since more folds are more computationally expensive [27]. Appendix E includes experiments with different values of k, with the most suitable option being a value of 10 which offers good performance without drastically increasing training time. K fold cross-validation is applied as part of the Reconstructed-DAE.ipynb notebook,

The loss function used is Mean Squared Error (MSE) [28], computed between the output of the model and the original trajectory. Roy and Bilodeau justifiably chose this loss function in [5], which is commonly used in sequence-to-sequence networks [12]. Another option here would be Binary Cross Entropy Loss, as seen in [2], which tends to have a higher learning rate, but sufferers more from overfitting, which is the problem we see the most in this approach. Appendix F includes my experiments with both loss functions, and the results justify the use of the MSE function, though both options offer good performance.

The optimizer differs from the one used in [5], as I have opted to go for the more traditional Adam [29], [30] optimizer, as opposed to RMSProp [30]. The decision to use RMSProp is not justified by Roy and Bilodeau in [5], and Adam has consistently provided better results in my tests (see

Appendix G). In fact, this optimizer is more utilized across similar literature, with [4], [12] choosing to opt for it instead of other options.

The reconstruction threshold is then calculated using the following formula, as suggested in [1].

$$mean(S_{tr}) + mean(S_{va}) + 3 * (STD(S_{tr}) + STD(S_{va}))$$

Where S_{tr} and S_{va} are vectors containing the MSE of each sample in the training and validation dataset respectively, and STD(x) is the standard deviation of an array x

Training happens over only 10 epochs, but since cross-validation is applied, each epoch has to be run 10 times such that each fold is used for validation once. I've kept the batch size to a relatively small power of two, as bigger batch sizes could lead to the model getting caught in local minima [31]. A condensed representation of all training hyperparameters can be seen in Table 3.1

Parameter	Value
Batch size	128
Epochs	10
Folds	10
Loss	Mean Squared Error
Optimiser	Adam
Learning Rate	0.001

Table 3.1: Reconstructed-DAE training parameters

3.2.4 Reconstructed-DAE architecture

The neural network architecture consists of fully connected linear layers. The encoder takes as input the original trajectory data consisting of a vector of length 125 and compresses it through 5 hidden layers, with the CFV consisting of only 8 elements. It's interesting to note that the first hidden layer increases the dimension of the data, and this is mainly due to the convenience of having a number of features that is a power of two, but it also helps the network learn more characteristics about the data in the encoding process. The decoder then takes the CFV through another 4 hidden layers which are symmetrical in size to the ones in the encoder, with the last layer representing the output layer, which matches the input in size. The output of the decoder is then compared to the inputted trajectory to calculate the Mean Squared Error loss.

The activation function between each hidden layer is the rectified linear unit (ReLU) function since it performs best in creating non-linearity [32]. After

the last layer of the decoder, the sigmoid function is applied to scale the data within the range of 0 and 1, just like the original data.

Figure 3.7 shows the Reconstructed-DAE architecture, including the dimensions of the data at each layer, and the activation functions used. It is interesting to note the similarities to other autoencoders used in literature, specifically in Figure 2.3, where we see a very similar diagram used by M. Ribeiro in [11], whose implementation used convolutional layers instead of linear layers.

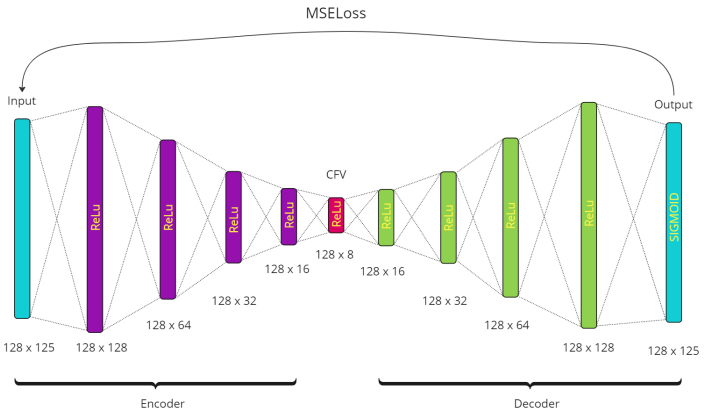

Figure 3.7: Diagram showing the Reconstructed-DAE architecture, where the most interesting thing to note is the input and the output of the network, which have to correspond in dimensions after the data has been compressed and decompressed while passing through the hidden layers. The process goes from left to right, with the encoder first creating a low-dimensional representation of the data (the CFV), which the decoder uses to recreate a trajectory. The more accurate the CFV is, and the better the decoder gets at interpreting it, the lower the Mean Square Error between the input and the output will get.

3.3 Second approach (Two-Input-RNN)

3.3.1 Long-Short-Term-Memory cells

LSTM networks [16], composed of LSTM cells, are a type of Recurrent Neural Network (RNN) [19] that was designed to fix the problem of vanishing gradients. These networks, like most other RNNs, excel at dealing with sequential data, but unlike other architectures, LSTM cells allow for information to be remembered or forgotten over time. In short, this means that only

very relevant information at the start of a sequence will be remembered, and used in the calculation of gradients later on in the sequence.

This architecture could prove very efficient in dealing with trajectory data, as Freitas showed in [1], where he used it for classifying trajectories. Trajectory data is sequential in nature, as each position of an agent is based on the previous position, and this information can be crucial in making decisions. In addition, other characteristics of the trajectory, like velocity, are also sequential (the speed at timestep n is heavily dependent on the speed at timestep n-1), so multi-dimensional data can be passed into the network effectively. For this reason, I have chosen to take an LSTM network approach to demonstrate the effectiveness of RNNs in detecting abnormal trajectories.

3.3.2 RNNs for abnormal trajectory detection

The process of detecting abnormal trajectories with an RNN follows a supervised learning approach for binary classification. First, the trajectory data is labeled as normal (1) or abnormal (0) and split into training, validation, and testing. The model is trained to take trajectory as input and output a single value, between 0 and 1, which represents the confidence that the given trajectory was normal. If the output is lower than a given threshold, the trajectory is classed as abnormal. Such a workflow can be seen in Figure 3.8 where a threshold of 0.5 is used. All of these steps, and the result generation and analysis are done in the Two-Input-RNN.ipynb jupyter notebook included in the code folder. For the creation of this model, the Tensorflow [33] framework was used.

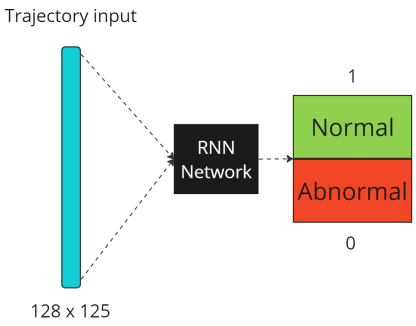

Figure 3.8: Trajectory data is passed into the network, which outputs a value between 0 and 1. A value greater than 0.5 means that the trajectory was normal, while a value lower than 0.5 means that the trajectory is abnormal.

3.3.3 Data preprocessing

A big concern with this approach is the dependency on a high volume of a diverse, inclusive dataset, with a similar number of normal and abnormal entries. However, as seen in Figure 3.4, the current dataset suffers from a high discrepancy between the two classes of data. This is not a problem for the autoencoder approach, however, balancing the dataset is crucial for this supervised learning approach.

To counter this problem, I apply oversampling on the training data. This means evenly multiplying every abnormal entry until their number matches the number of normal entries. Even if this is not as good as an evenly distributed dataset of 'real' trajectories, it avoids the issue of overfitting the network to a single class [34]. It is important to notice here that only training data is oversampled, while validation and testing data remain unbalanced to fairly evaluate the performance of the network. Undersampling the over-represented class was also a possibility, but due to the small size of the dataset, I chose against that technique. Figure 3.9 shows the results of the class balancing technique on the Sherbrooke dataset.

Figure 3.9: After oversampling, the training dataset is balanced (equal number of normal and abnormal entries). However, testing and validation datasets are kept unbalanced, since those are used to evaluate the performance of the network.

Another issue with the dataset is represented by the categorical feature representing the type of vehicle that created the trajectory. Originally, this feature is represented by a 0 for a pedestrian, 1 for a car, and 2 for a bicycle. However, this ordinal encoding of categorical data is proven to be detrimental to networks' ability to learn [35]. There has been a lot of research on the subject of encoding categorical features [36], but for the purpose of this project, there are two main options. One is to have encoding layers preprocess categorical data in the input before passing it forward

into the network. While this approach is technically the most viable one, it is not required for a single categorical feature with three classes. Instead, I have chosen to apply one-hot encoding, which transforms the classes into a unique binary array with a single value of one, representing the class.

The last step before the data is ready to be used in the RNN is reshaping it. Originally the dataset is represented by vectors of size 125 following the format [id, category, x1, y1, vx1, vy1, x2, y2, vx2, vy2, ..., x31, y31, vx31, vy31]. While this representation was appropriate for the deep autoencoder method, it does not actually maintain the sequential nature of trajectories. Instead, I have now reshaped the data to a vector of shape [4, 31], where the first axis represents the 4 features (x position, y position, x velocity, y velocity] and the second axis represents the value of that feature at one of 31 timesteps. In addition, the ID is dropped as it has no relevance. Now the data is composed of multi-featured sequential trajectories which work best for RNN architectures. The transformation of the data can be seen in Figure 3.10. It is interesting to mention the similarities to the input format in Figure 2.4, where Freitas [1] aims to make use of the sequential nature of trajectory features, demonstrating the benefits of large multi-featured sequential data.

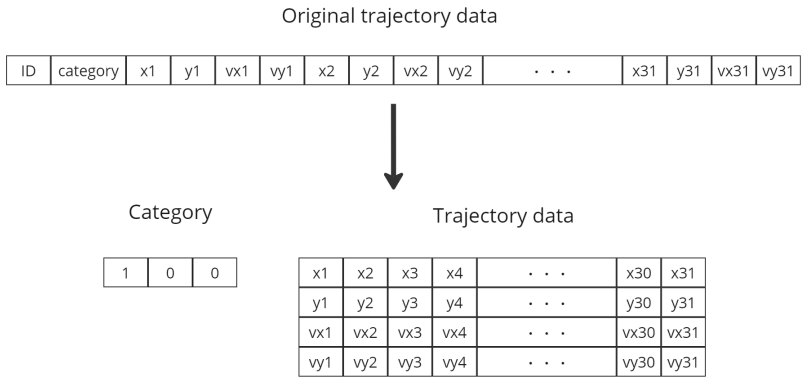

Figure 3.10: The original trajectory data is restructured into a 2-dimensional array, representing four sequential features of a trajectory (x and y coordinates, and velocities along the x and y axis) across 31 timesteps. The categorical data is one-hot encoded and saved into a different data structure.

3.3.4 Two-Input-RNN architecture with LSTM cells

This novel architecture takes advantage of TensorFlow's multiple-input models to create a network that takes two inputs. The trajectory data goes into the LSTM cell which learns the characteristics of the trajectory without

being affected by the categorical data that doesn't follow a sequential logic and outputs a vector of size 100. This output is then concatenated with the second input of the model, the category of the trajectory, and passed into a dense layer that uses a sigmoid activation function to output a value between 0 and 1 representing the binary classification of the input. Figure 3.11 shows a diagram representing this novel architecture.

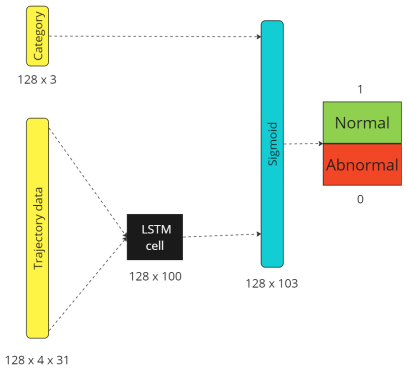

Figure 3.11: The novelty of this architecture lies in the multiple-input model. One input is the trajectory data which is passed to the LSTM cells, while the other input is the categorical data, which doesn't follow a sequential logic and so it skips the LSTM layer. The output of the LSTM layer and the categorical data are concatenated and passed into a linear layer with a sigmoid activation function, which classifies the trajectory.

The loss function used in training is Binary Cross Entropy (BCEL) since it is often the best option for binary classification problems [37], and unlike the deep autoencoder method, we are not dealing with sequence-to-sequence networks anymore, where MSE [28] was the better choice, but a sequence-to-one network (see Appendix H for experiments). The optimizer is once again Adam [29] with a learning rate of 0.001, as this is the most common choice among similar papers, like [4] and [12]. and experiments show better performance compared to RMSProp once again (see Appendix I). Training takes place over only 20 epochs, as the model quickly converges to a very good solution. Table 3.2 shows a compressed view of all hyperparameters chosen for this model.

Parameter	Value
Batch size	128
Epochs	20
Loss	Binary Cross Entropy
Optimiser	Adam
Learning Rate	0.001

Table 3.2: Two-Input-RNN training hyperparameters

3.4 Third approach (DBSCAN-RNN)

3.4.1 Clustering unlabelled data

The unsupervised training approach used by the Reconstructed-DAE had the benefit that data didn't have to be labeled before training, which is a significant advantage when considering the deployment of such a system in a real-world scenario. Supervised learning models like the Two-Input-RNN however require the entire dataset to be labeled, and doing so manually would be inefficient. Instead, I propose an automated way of labeling the data before training, and based on [2], I have chosen to use the DBSCAN [10] clustering algorithm to create a semi-supervised approach. This algorithm assigns each trajectory to a cluster or marks it as "noise" if it doesn't fit in any of the existing clusters. Therefore, in my implementation, every trajectory that is assigned to a cluster is considered normal, while the outliers (or "noise") consist of abnormal data. Figure 3.12 shows how the DBSCAN algorithm works to label the dataset. The labels obtained through this method are not 100% accurate but do not require human intervention. The results of such a partially correct dataset as opposed to a fully labeled dataset will be explored in future sections.

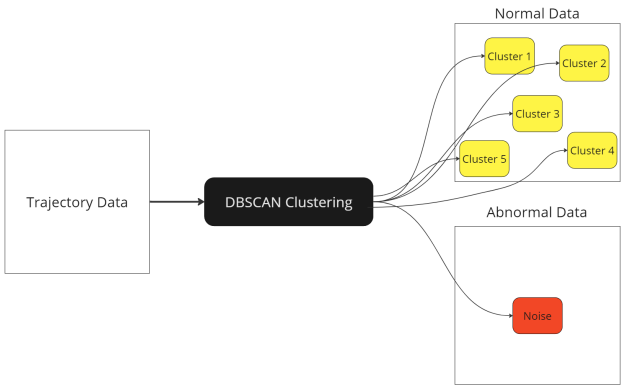

Figure 3.12: The DBSCAN algorithm takes the trajectory data as input and outputs n clusters of trajectories and "noise" samples that did not fit in any of the clusters. The clustered data is then labeled as normal while the noise is labeled as abnormal.

3.4.2 DBSCAN-RNN workflow

The workflow now consists of one extra step compared to the Two-Input-RNN approach. After the data is split into training, validation, and testing like

in the second method, the training dataset is now labeled by the DBSCAN clustering algorithm, while the validation and testing sets, which are a lot smaller are manually labeled. The same Two-Input-RNN model is then trained and tested using these datasets. This means that the network will train on a partially incorrect dataset, as DBSCAN won't have perfect accuracy when labeling, but the validation and testing datasets are always 100% accurate. While this will reduce the performance of the model, it will drastically reduce the amount of human intervention needed in deployment. Figure 3.13 shows this workflow in detail. All of these steps, and the result generation and analysis are done in the DBSCAN-RNN.ipynb jupyter notebook included in the code folder. The same Tensoflow [33] framework was used here as well.

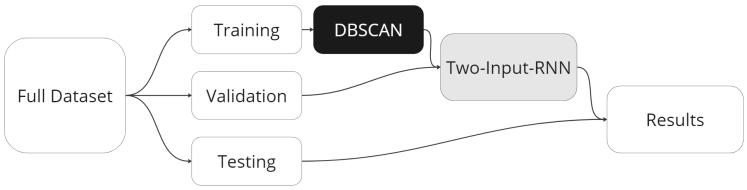

Figure 3.13: After splitting the dataset in training, validation, and testing, DBSCAN is used to label the training data, while the other two subsets are labeled "manually". The Two-Input-RNN learns from the training data and then results are generated via the testing dataset.

4 Results and analysis

4.1 Testing methodology and metrics

The results are generated by testing the Reconstructed-DAE, the Two-Input-RNN, and the DBSCAN-RNN methods on the four datasets presented in section 3.1, after creating the networks described in the methodology section.

The Machine learning field has a large plateau of metrics available for measuring a model's performance. In this project, I will focus on accuracy, measured in x% where x is the percentage of trajectory correctly classified

as abnormal/normal. Because this is a problem of binary classification, the confusion matrix is a valuable metric that I will be using to analyze predictions. The confusion matrix is a 2x2 matrix comprising the true positive, false positive, true negative, and false negative metrics. The final loss value of a model, or various iterations of it from training will be used whenever relevant. Results were generated using the Reconstructed-DAE.ipynb, Two-Input-RNN.ipynb, and DBSCAN-RNN.ipynb jupyter notebooks respectively, all of which are included in the code folder.

4.2 Reconstructed-DAE results

The training aspect of this model is particularly interesting to observe. The reconstruction error on the normal data in the early stages is very high, which causes the threshold value to also be high. A high threshold means that the model will assume every trajectory is a normal trajectory, even though it doesn't fully understand what a normal trajectory looks like. The high threshold causes the testing accuracy on the abnormal data to be very low. Iteration after iteration, as the network learns to better reconstruct trajectories, the threshold becomes lower, and so the accuracy of the abnormal data increases. Training then becomes a game of bringing the threshold as low as possible, such that every normal trajectory is still classified as normal, but as many abnormal trajectories are reconstructed with a value larger than this threshold. The key to balancing this act of course lies in how well the autoencoder can reconstruct trajectories, as those values influence how low the threshold gets, and how high the abnormal data reconstruction is.

The plots in Figure 4.1 show this clear correlation between threshold and testing accuracy during training on the Sherbrooke dataset. It is also clear due to the slow, constant increase in accuracy that the model first finds the most obviously abnormal trajectories, and as it slowly gets better it learns to detect the harder examples too. As the abnormal trajectories get more and more similar to the normal ones, the network starts to struggle. However, another interesting aspect is the accuracy on the normal data. This value never really drops below the 99.9% mark, which means that the model is doing a great job at maintaining the threshold just large enough to cover almost the entire normal data at all times. This can be observed in the plot in Figure 4.2. Appendix J contains similar results for the other three datasets.

Now let's look at some of the final results achieved by this model. Figure 4.3 shows the confusion matrix after testing on each of the four datasets. The most notable aspect, which is in line with the observations in training, is that

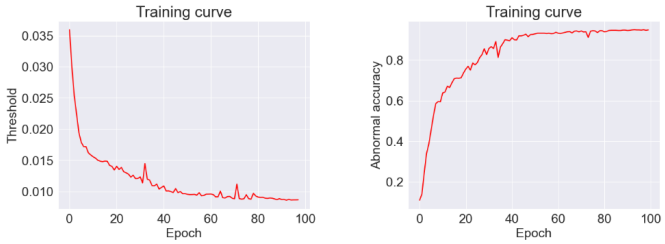

Figure 4.1: On the left, the evolution of the threshold value across the 100 iterations, on the right the accuracy obtained by the model on the abnormal data at every one of those iterations. The correlation between the two is clear, even in the spikes around epoch 35 and epoch 75.

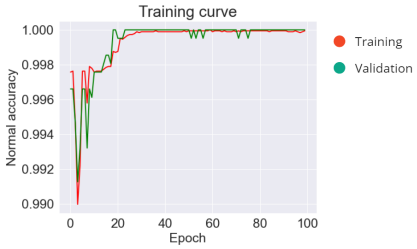

Figure 4.2: This plot shows that the accuracy of the model on the normal data never drops under 99%. In fact, the lowest accuracy is obtained at the start of the training process, but after a few iterations, the model maximizes its accuracy on the training data while still reducing the threshold value.

the model tends to classify most trajectories as normal, resulting in very high normal accuracy, but low abnormal accuracy, especially on the Rouen and St-Marc datasets. A comparison to the results in [5] can be seen in Table 4.1, and the difference in accuracy can be attributed to training time, the change in the optimizer, the application of k-fold cross-validation or other hyperparameter differences that are not clearly described in Roy and Bilodeau's implementation [5].

	sherbrooke	rene	rouen	stmarc
Model	Abnormal|Normal	Abnormal|Normal	Abnormal|Normal	Abnormal|Normal
Original DAE	80|99	85|100	32|99	39|99
Reconstructed-DAE	75|99	87|99	28|100	12|99

Table 4.1: The results on the normal data are in line with that obtained by Roy and Bilodeau [5], however, accuracy on the abnormal data is a few percent lower across all datasets.

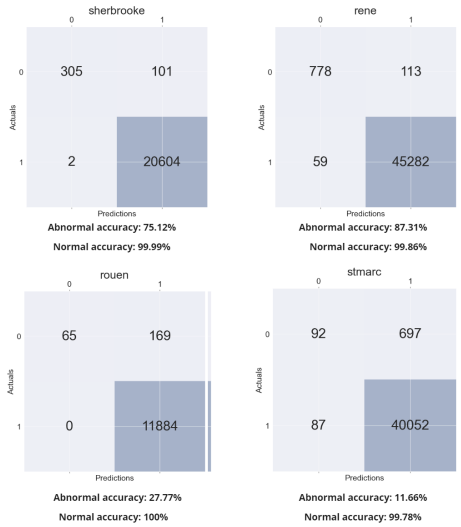

Figure 4.3: Reconstructed-DAE method results displayed using a confusion matrix for each dataset. The model correctly identifies most normal trajectories but the number of false positives is high, resulting in a low abnormal accuracy.

4.3 Two-Input-RNN results

Training is done on 80% of the data, of which 20% is only used for validation, while testing is done on the rest of 20% of data. The results shown are the accuracy obtained on the testing data, which are the trajectories the model doesn't see during training. This subset is smaller than the set of trajectories that were used in testing the Reconstructed-DAE, but that doesn't invalidate the results.

While the architecture of this network is relatively small, the results of this supervised learning approach are outstanding in comparison to the deep autoencoder method. This network is capable of correctly classifying all normal and abnormal trajectories in all four datasets after only a few epochs. Figure 4.4 shows the complete results on all datasets, and as suggested by the confusion matrix, the accuracy of this approach is 100% for three of the four datasets. This once again proves the amazing capacity of RNNs to interpret a large amount of sequential data, but this approach requires a high amount of manual intervention for labeling the dataset.

Looking at the training metrics, it can be observed that the network is having an easy time classifying the trajectories, as it converges to a solution relatively quickly. Figure 4.5 shows plots of the loss, the accuracy, and the perhaps more relevant false positive and false negative values during training on the rouen dataset. All of those suggest a slow and steady

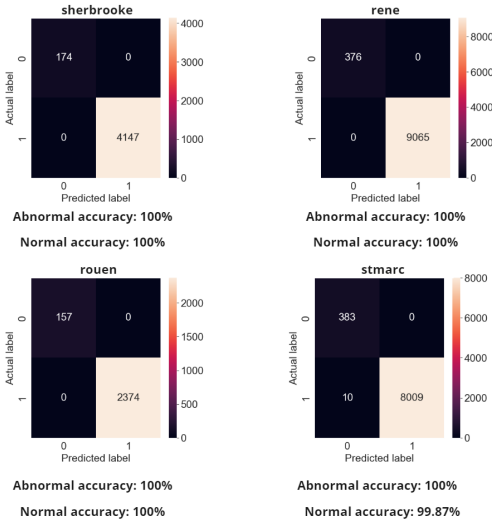

Figure 4.4: The Two-Input-RNN achieves almost perfect accuracy on all datasets, the only mistakes are on normal data in St-Marc, but abnormal trajectories are always classified correctly.

increase in performance until the model eventually converges to a final solution. Appendix K includes training statistics for all the other datasets.

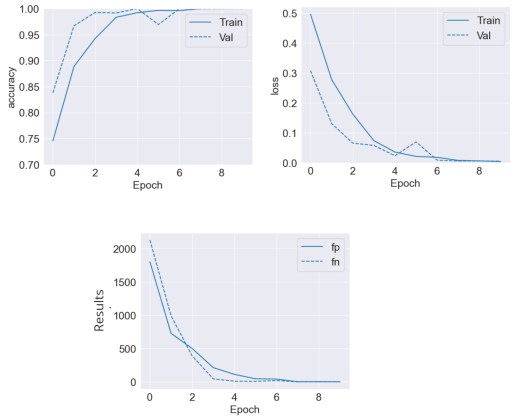

Figure 4.5: The static increase in accuracy corresponds with the decreasing number of false positives and false negatives in each iteration. The network is learning fast and reaches maximum accuracy before the 20th epoch.

4.4 DBSCAN-RNN results

When clustering data with DBSCAN [10], it is important to define how close a trajectory needs to be to an existing cluster in order to be considered part of it. The epsilon parameter (eps) [10] controls this aspect and in order to find the best eps value for each dataset, I have run experiments to determine which eps value offers the best results in labeling the dataset correctly. Figure 4.6 shows these experiments and the best eps value for each of the four datasets. Most noticeably, a high eps value classifies more of the data as normal, while a lower value classifies more of it as abnormal. A "mean" accuracy is used to determine which eps value is best suited.

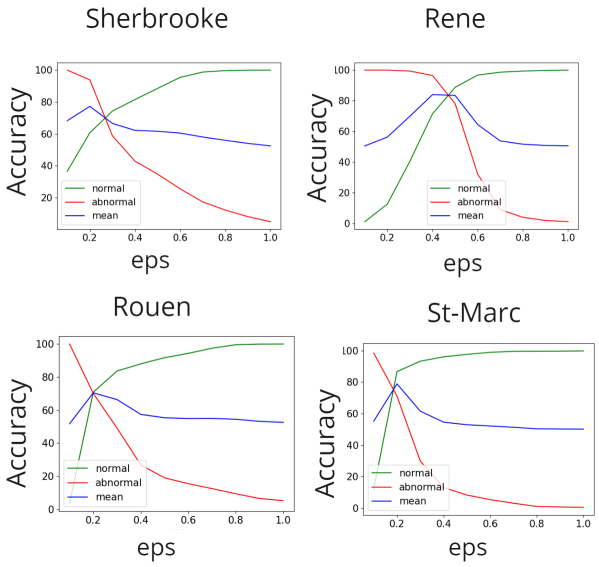

Figure 4.6: A eps value of 0.2 offers the best results for all datasets except for Rene, where maybe due to the higher amount of abnormal data, a value of 0.4 was the most effective instead. It is obvious how as the eps value increases the normal accuracy increases while the abnormal accuracy decreases.

The results of DBSCAN-RNN on each dataset can be seen in Figure 4.7, and while the accuracy has decreased with the addition of automatic labeling, the results are still competitive with those observed with the Reconstructed-DAE. The drop in normal accuracy is very significant here, but it's important to note that a different eps value for DBSCAN would produce higher normal accuracy at the cost of abnormal accuracy. This offers users the possibility to adapt their model's performance based on their needs, which could be a neat feature in a real-world system like this. The results displayed are computed such that the mean accuracy is as big as possible.

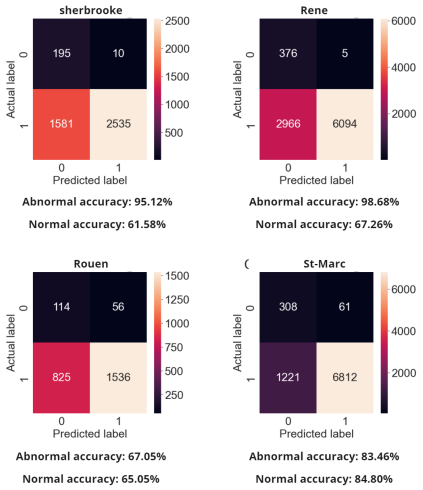

Figure 4.7: When using the DBSCAN algorithm to cluster the data before training, the performance of the RNN is good, but not perfect. In this scenario, abnormal accuracy performs very well, and better than in the Reconstructed-DAE method, but the normal accuracy has dropped significantly.

It is interesting to note here that the accuracy of DBSCAN-RNN is better than that of DBSCAN alone, but it seems to be the case that the clustering accuracy of DBSCAN is the limiting factor for the model's accuracy, as seen in Table 4.2. It can be deduced that the model is perfectly capable of learning from the training data, however, if the training data is not perfectly labeled then the end results will be affected. It is also important to note that the comparison between DBSCAN and DBSCAN-RNN is only done in order to justify the reduced accuracy of DBSCAN-RNN, as DBSCAN is not capable of abnormal trajectory detection in real-time, and it's only used for training purposes for DBSCAN-RNN.

	sherbrooke	rene	rouen	stmarc
Model	Abnormal|Normal	Abnormal|Normal	Abnormal|Normal	Abnormal|Normal
DBSCAN	93|60	96|71	70|70	71|86
DBSCAN+RNN	61|95	98|67	67|65	83|84

Table 4.2: The DBSCAN-RNN method performs better than DBSCAN alone in most cases, but the inaccuracy of the clustering algorithm is the reason why the RNN doesn't achieve better results.

Looking at the training metrics for the DBSCAN-RNN network on the Rouen dataset in Figure 4.8, the training accuracy and loss suggest that the network is having an easy time learning to classify these trajectories, as it converges to a solution quickly. However, on the validation dataset, these results are not mirrored, and this is due to the training dataset not being 100% accurate in labels. What is happening is that the network is doing a good job learning from the training data, but the training data is not perfect,

so neither will the testing results. The high accuracy in training however suggests that given a perfectly labeled dataset, this network could achieve near-perfect results, which was proved to be true in section 4.3. Appendix L includes the training metrics for all the other datasets too.

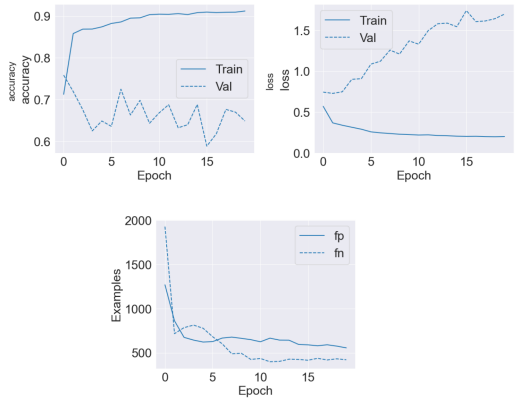

Figure 4.8: The lack of consistency between accuracy and loss in both training and validation suggests that the network is learning what the training dataset has to offer, but that may not be in line with the validation dataset. This suggests that the network is limited not by its ability to learn, but by the quality of the data.

4.5 Creating custom dataset

To delve deeper into the pros and cons of each model, and to take a better look at the particular trajectories that are not getting classified correctly, I have created a new dataset, using an intersection in York, where I've generated normal and abnormal trajectories. This can give us a better insight into what's stopping the Reconstructed-DAE to perform as well as the RNNs, and what type of entries it struggles with the most.

4.5.1 York dataset

I have created this dataset manually in an attempt to provide difficult trajectories for the models. This dataset consists of 8200 normal trajectories and 1620 abnormal trajectories, categorized as pedestrian, vehicle, or bicycle. Only a handful of those trajectories are manually created. These original trajectories are then multiplied by applying a randomized element-wise addition, following a normal distribution. There are 82 original normal data points that are multiplied by 100, and 162 original abnormal data points

that are multiplied by 10. The multiplication through a normal distribution of variance guarantees some level of diversity in the dataset, and it somewhat simulates a real dataset. Compared to the trajectories in the other datasets, these trajectories do not include velocity data and are instead only composed of spatial coordinates. In addition, each sub-trajectory is composed of 50 timesteps instead of 31, so the total length of the data array is now 102. Appendix M shows the structure of such a trajectory, compared to the previous data. The original entries are created using the GeoGebra platform [38]. Figure 4.8 shows the original data in the York dataset, generated using the view_trajectory.py script, written by the author of this project and included in the code folder.

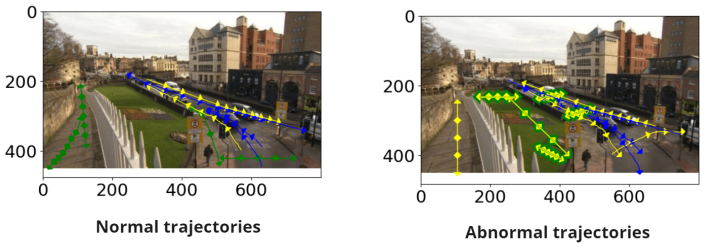

Figure 4.9: York dataset. Green for pedestrians, blue for vehicles, and yellow for bicycles

4.5.2 York dataset results

When comparing the three models, it's important to note the difference in the dimensions of the testing data. The Reconstructed-DAE is tested on all the normal and abnormal trajectories, but the RNNs can only be tested on trajectories that weren't used in training, so it's a smaller subset. However, the accuracies on this dataset confirm what we've been seeing in previous results too. The Reconstructed-DAE performs very well on normal data, while the Two-Input-RNN and the DBSCAN-RNN are better suited for detecting anomalies. Overall however not even this dataset presents a bigger challenge for the Two-Input-RNN, which proves to be the highest performance model once again, but the performance of DBSCAN-RNN is very high compared to the other four datasets. Figure 4.10 shows the confusion matrix result for each method on this dataset.

Figure 4.11 shows the trajectories that were misclassified by each of the three networks on the York dataset. This data shows that all the networks are able of distinguishing between the different types of agents in the scene. This reassuring fact is confirmed by the lack of misclassified abnormal trajectories like bikes riding on the city walls or on the grass, which is exactly what I was testing for with this dataset. The Two-Input-RNN and the

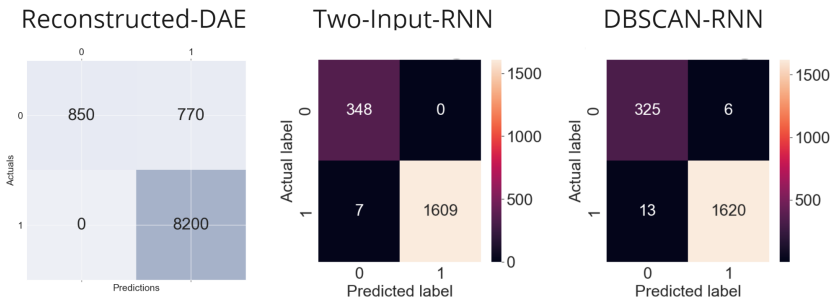

Figure 4.10: The Reconstructed-DAE is the only model to obtain 100% accuracy on the normal dataset, but the RNNs perform better on abnormal data. DBSCAN-RNN's performance is almost identical to Two-Input-RNN's, which is not what we've seen in previous results.

DBSCAN-RNN make similar mistakes, with the notable difference being that DBSCAN-RNN makes a few more, and some of those mistakes are on abnormal data. More interestingly however is that while the Reconstructed-DAE makes significantly more mistakes, we can see that some of the mistakes of the RNN architectures are not made by the DAE, specifically the pedestrians walking on the walls being classified as abnormal by both RNNs. This once again proves the value of this network for scenarios when the correct classification of normal data is of utmost importance.

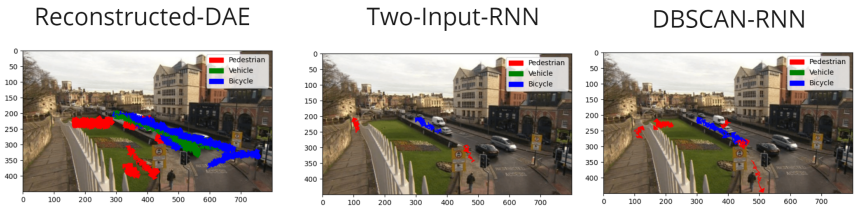

Figure 4.11: The misclassified trajectories of each network on the York dataset. The Reconstructed-DAE is the only one to misclassify vehicle trajectories. The mistakes of the Two-Input-RNN seem to be a subset of the DBSCAN-RNN's mistakes.

4.6 Final analysis and comparison

Table 4.3 shows the final results of the three models on all of the four initial datasets, and the York dataset. Overall, the best model in terms of accuracy is the Two-Input-RNN. The Reconstructed-DAE that requires less human

intervention offers low accuracy on abnormal data but performs excellently on normal data. The DBSCAN-RNN is a middle-ground approach that compromises between accuracy and ease of deployability, but it shows to be more dependent on the spread of trajectories in the scene. To compare results with other state-of-the-art models on the original datasets, I've added the results of the original DAE in [5] and the extended version of that implementation using a Generative Adversarial Network in [6], called ALREC (Adversarially Learned Reconstruction Error Classifier).

	sherbrooke	rene	rouen	stmarc	York
Model	Abnormal|Normal	Abnormal|Normal	Abnormal|Normal	Abnormal|Normal	
Reconstructed-DAE	75|99	87|99	28|100	12|99	52|100
Two-Input-RNN	100|100	100|100	100|100	100|99	100|99
DBSCAN-RNN	95|64	98|67	67|65	83|84	98|99
Original DAE	80|99	85|100	32|99	39|99	?|?
ALREC	96|99	96|100	92|99	62|99	?|?

Table 4.3: Final results on all five datasets, expressed in accuracy percentage. The novel architecture Two-Input-RNN and DBSCAN-RNN can compete with ALREC in abnormal data, but only the Two-Input-RNN is also competitive in normal data. However, the Reconstructed-DAE is only competitive in normal data. The original DAE and ALREC were not tested on the York dataset.

Ultimately, the decision of which method performs best depends on multiple factors, and a user would have to decide on the best-suited implementation for a particular case. For example, the Reconstructed-DAE may be favored when the misclassification of normal trajectories poses a greater risk than the misclassification of abnormal trajectories, and vice-versa for the Two-Input-RNN. There is also the factor of how much investing in human resources is available. The Reconstructed-DAE requires no manual labeling of the dataset, the DBSCAN-RNN requires minimal labeling for the validation and testing datasets, while the Two-Input-RNN requires a much higher human intervention to function. Overall these three methods cover a lot of the possible use cases, as they range from cheap to expensive and from lower performance to higher performance.

5 Conclusion and future work

In conclusion, the problem of detecting abnormal trajectories can be solved in a multitude of ways. Approaches using autoencoders [5], [11], generative adversarial networks [6], recurrent neural networks [1], [12], [18], image classification [20], and data clustering [2] have all been tested throughout time, as this challenge has been thoroughly explored. The three methods

for solving abnormal trajectory detection that I present (Reconstructed-DAE, Two-Input-RNN, and DBSCAN-RNN) only scratch the surface of the seemingly endless ways of solving this challenge.

However, from a deep-learning perspective, a lot of ground is covered in this report, as I've dived deep into the details of deep autoencoders [24] and recurrent neural networks [19]. Through testing and experimentation, this study identified optimal training hyperparameters and techniques for DAE architectures, RNN architectures using LSTM cells [16], and combinations of clustering algorithms and the same RNN networks. In addition, the distinctions among supervised, semi-supervised, and unsupervised learning have been examined with the intention of facilitating real-world applications.

With the idea to create an efficient model for taking advantage of the sequential nature of trajectory data, I've proposed a novel architecture for an RNN with two inputs, which can take sequential and non-sequential data as input, something that wasn't explored in any of the previous literature. The two networks that were built on these foundations, the Two-Input-RNN and the DBSCAN-RNN have proven to be promising by obtaining state-of-the-art results and outperforming the models in [5] and [6] on the same datasets [22]. The testing should be continued on other datasets and against other state-of-the-art models because as seen in this report, any dataset can be processed into a suitable structure for the testing of these implementations. It would also be valuable to explore the performance of these methods on classification problems, as opposed to abnormal trajectory detection, as these networks are capable of classifying a trajectory to any of n classes, and are used in this project only as binary classifiers.

For future work, it would be most interesting to further delve into the power of RNNs, perhaps by implementing a fully unsupervised learning approach that uses the LSTM cells for learning. In such an approach, an architecture similar to that of the Reconstructed-DAE would be used, but this time instead of linear layers compressing the data, LSTM sequential-to-sequential layers would be used instead. Such an implementation would give a clear comparison between LSTM cells and simple linear layers, and the advantages would perhaps be easier to observe. It would also be valuable to delve deeper into convolutional autoencoders and how well those can compete with RNNs, as they do not take advantage of sequential data, but they've proven very efficient in [11].

Part III

Appendix

A Abnormal data view

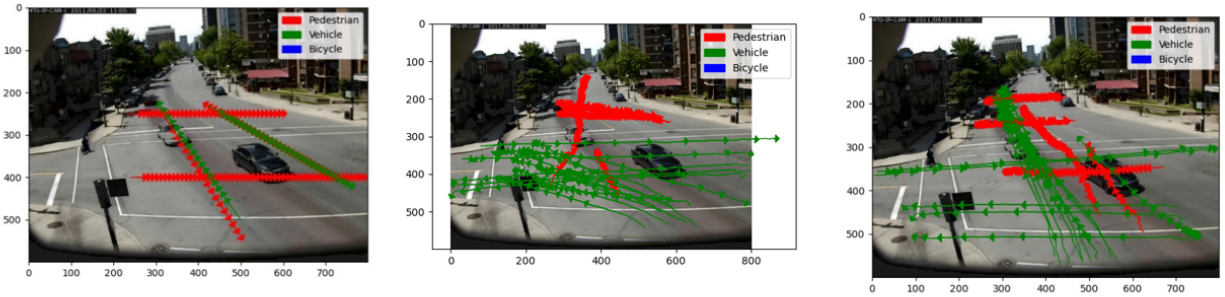

Figure 1: Sherbrooke abnormal data

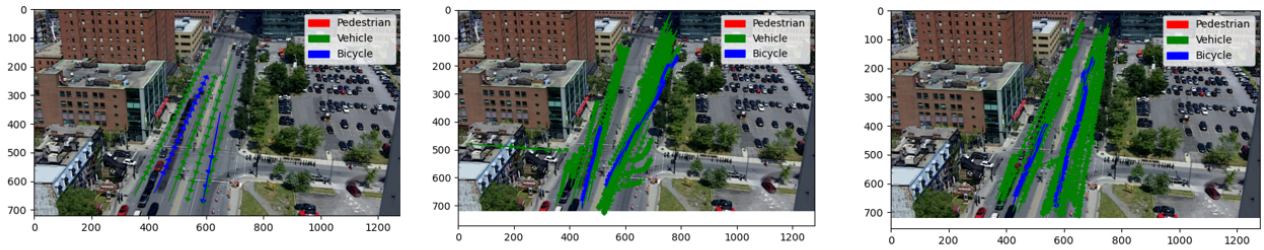

Figure 2: Rene abnormal data

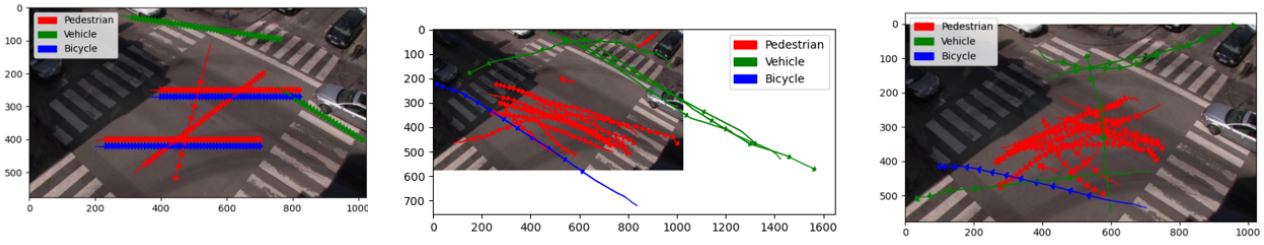

Figure 3: Rouen abnormal data

Figure A.1: Three figures showing abnormal data in the Sherbrooke, Rene, and Rouen datasets

B Sub-trajectories

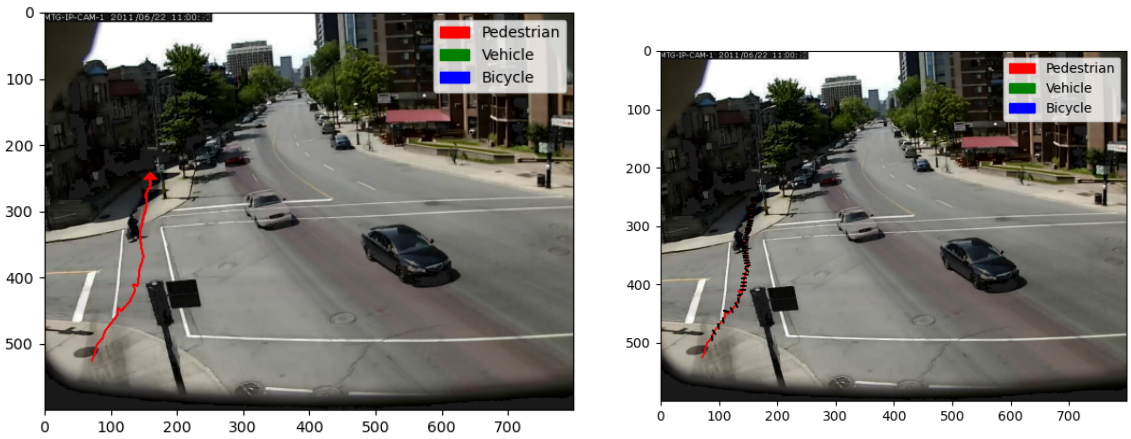

Figure B.1: On the left, a full trajectory collected from the point the agent is identified in the scene until it is no longer seen. On the right, the same trajectory after it's been split into fixed-length sub-trajectories. The black dividers represent where sub-trajectories end.

C Data format

Original trajectory data

ID	category	x1	y1	vx1	vy1	x2	y2	vx2	vy2	. . .	x31	y31	vx31	vy31

Figure C.1: Data format in the four original datasets, where x and y represent the spatial coordinates at each of the 31 timesteps, vx and vy represent the velocities at each of the 31 timesteps, category represents the type of trajectory for pedestrian, bicycle or vehicle, and ID represents the ID of the trajectory that the sub-trajectory belongs to.

D Data normalisation

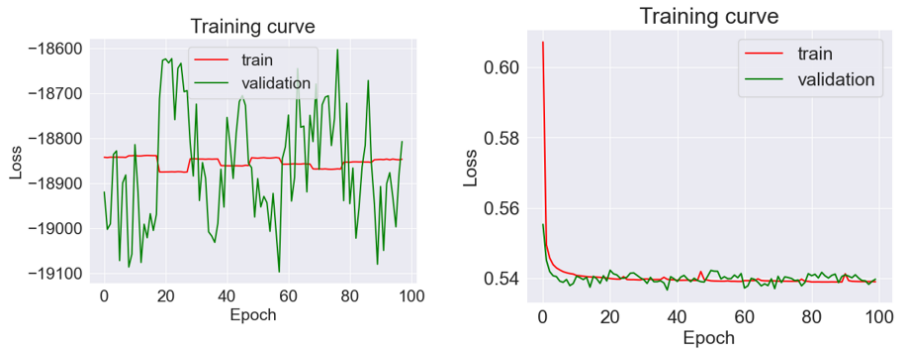

Figure D.1: Effects of normalization on the training performance of the network. Without normalization (left plot) the model is not learning since the loss is not decreasing with each epoch. On the right, however, when data is normalized, the loss function decreases so the network is learning. Experiments were conducted on the Sherbrooke data.

E K-fold cross-validation

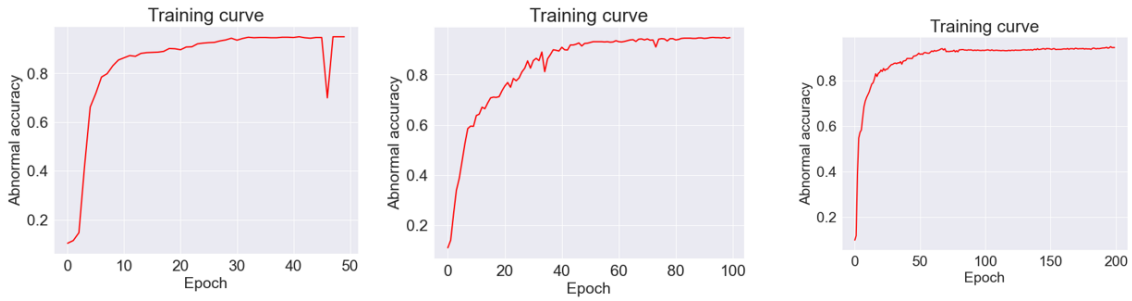

Figure E.1: Accuracy of models using cross validation with k=5 (left), k=10 (middle), k=20 (right). Results are similar but training with 20 folds takes significantly more time. Experiments were conducted on the Sherbrooke data.

F MSE vs BCEL

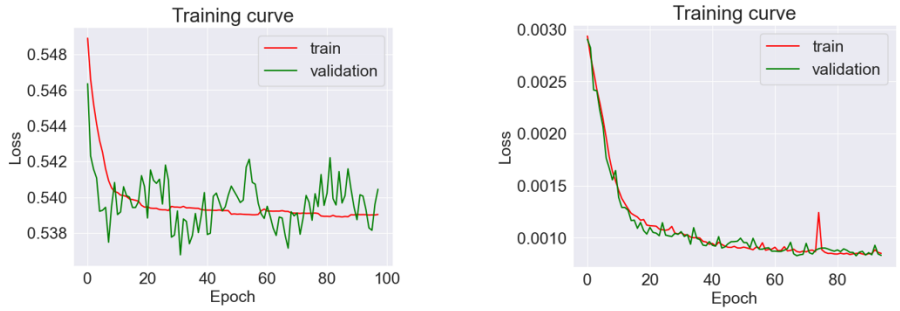

Figure F.1: Mean Square Error loss (right) produces a smoother curve for the loss function, compared to Binary Cross Entropy loss (left), which produces a lot of spikes, especially in validation. Results are similar but MSE is preferred. Experiments were conducted on the Sherbrooke data.

G RMSProp vs Adam

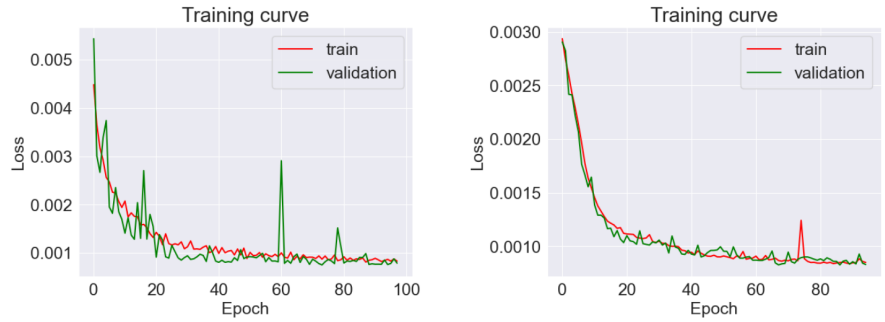

Figure G.1: Adam (right) produces a smoother curve for the loss function, compared to RMSProp loss (left), which produces a lot of spikes, especially in validation. Experiments were conducted on the Sherbrooke data.

H MSE vs BCEL in Two-Input-RNN

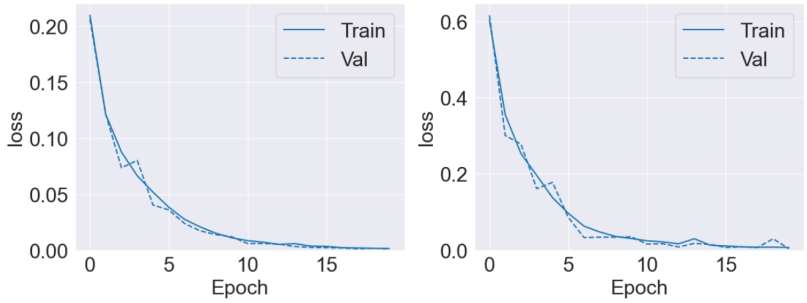

Figure H.1: Mean Square Error loss (right) and Binary Cross Entropy loss (left), produce almost identical results so the decision is not very relevant. Experiments were conducted on the Sherbrooke data.

I RMSProp vs Adam in Two-Input-RNN

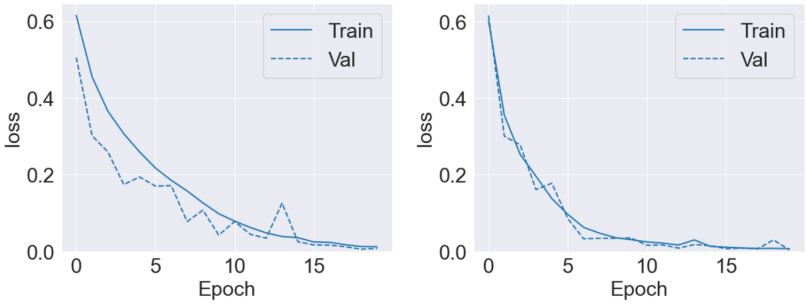

Figure I.1: Adam (right) produces a smoother curve for the loss function, compared to RMSProp loss (left), which produces a few more spikes, especially in validation. The final results however are very similar. Experiments were conducted on the Sherbrooke data.

J Reconstructed-DAE training results

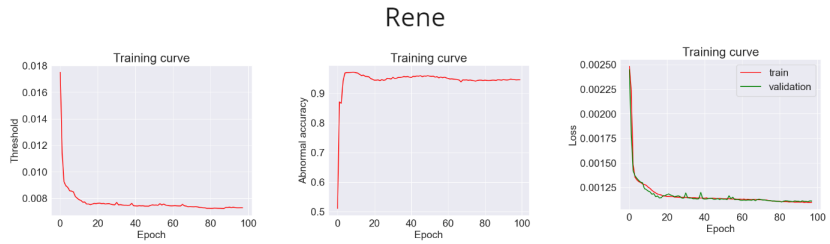

Figure J.1: Rene training results.

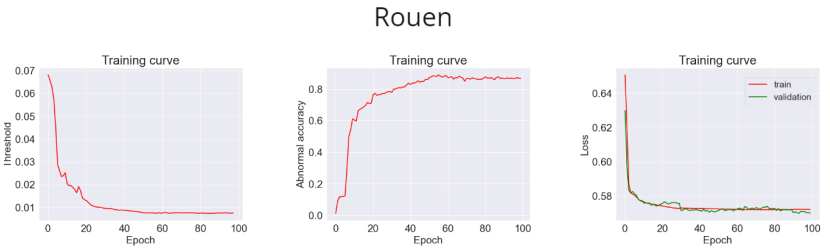

Figure J.2: Rouen training results.

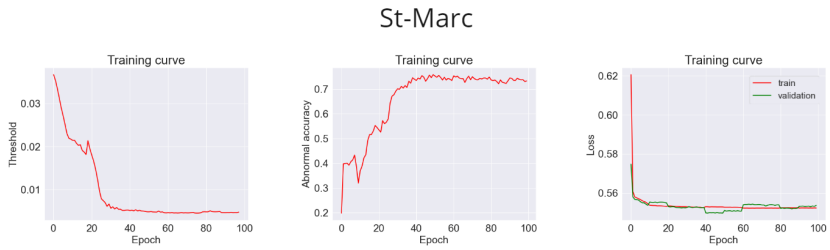

Figure J.3: St-Marc training results.

K Two-Input-RNN training results

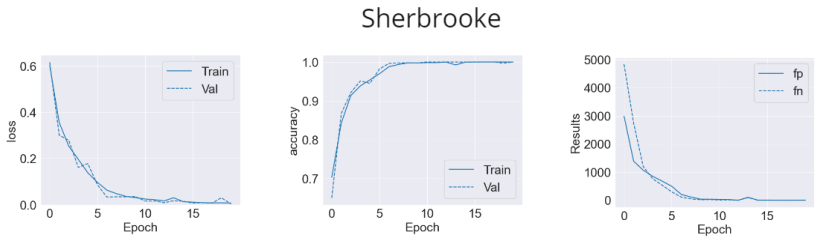

Figure K.1: Sherbrooke training results.

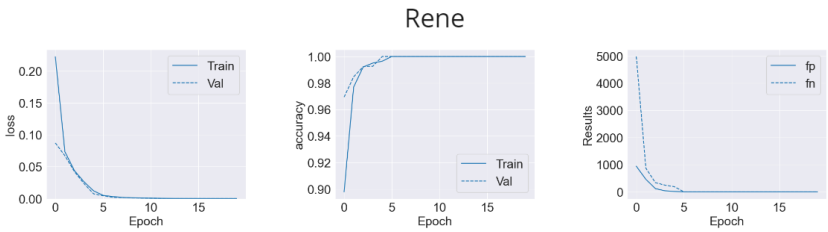

Figure K.2: Rene training results.

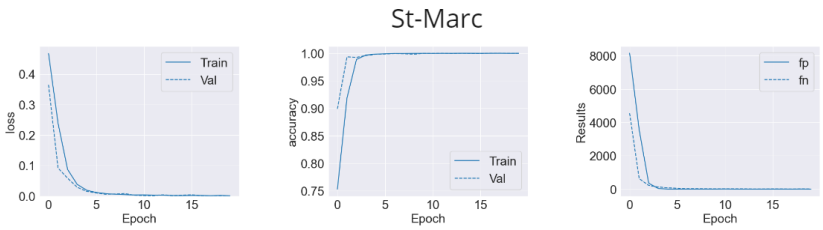

Figure K.3: St-Marc training results.

L DBSCAN-RNN training results

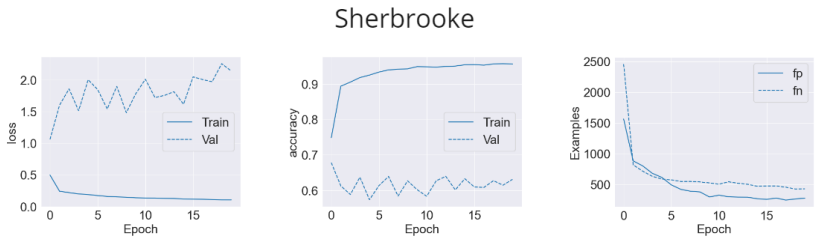

Figure L.1: Sherbrooke training results.

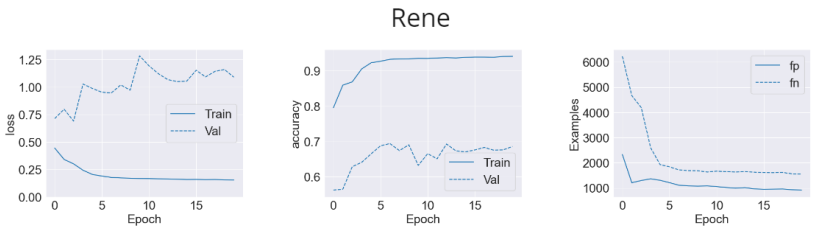

Figure L.2: Rene training results.

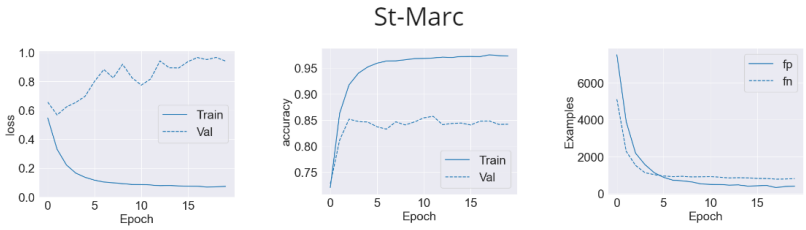

Figure L.3: St-Marc training results.

M York data format

York trajectory data

ID	category	x1	y1	x2	y2	x3	y3	x4	y4	. . .	x49	y49	x50	y50

Figure M.1: York data format, where x and y represent the spatial coordinates at each of the 50 timesteps, category represents the type of trajectory for pedestrian, bicycle or vehicle, and ID represents the ID of the trajectory that the sub-trajectory belongs to.

Bibliography

[1] N. C. de Freitas, T. L. C. da Silva, J. A. F. de Macêdo, L. M. Junior and M. G. Cordeiro, 'Using deep learning for trajectory classification,' in *Proceedings of the 13th International Conference on Agents and Artificial Intelligence (ICAART 2021)*, 2021.

[2] T. Zhang, S. Zhao, B. Cheng and J. Chen, 'Atedlw: Intelligent detection of abnormal trajectory in ship data service system,' in *2021 IEEE International Conference on Services Computing (SCC)*, IEEE, 2021, pp. 401–406.

[3] H. Duan, F. Ma, L. Miao and C. Zhang, 'A semi-supervised deep learning approach for vessel trajectory classification based on ais data,' *Ocean & Coastal Management*, vol. 218, p. 106 015, 2022.

[4] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese and A. Alahi, 'Social gan: Socially acceptable trajectories with generative adversarial networks,' in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2255–2264.

[5] P. R. Roy and G.-A. Bilodeau, 'Road user abnormal trajectory detection using a deep autoencoder,' in *Advances in Visual Computing: 13th International Symposium, ISVC 2018, Las Vegas, NV, USA, November 19–21, 2018, Proceedings 13*, Springer, 2018, pp. 748–757.

[6] P. R. Roy and G.-A. Bilodeau, 'Adversarially learned abnormal trajectory classifier,' in *2019 16th Conference on Computer and Robot Vision (CRV)*, IEEE, 2019, pp. 65–72.

[7] D. Samariya, J. Ma, S. Aryal and X. Zhao, 'Detection and explanation of anomalies in healthcare data,' *Health Information Science and Systems*, vol. 11, no. 1, p. 20, 2023.

[8] J. Bian, D. Tian, Y. Tang and D. Tao, 'A survey on trajectory clustering analysis,' *arXiv preprint arXiv:1802.06971*, 2018.

[9] F. T. Liu, K. M. Ting and Z.-H. Zhou, 'Isolation forest,' in *2008 eighth ieee international conference on data mining*, IEEE, 2008, pp. 413–422.

[10] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, 'A density-based algorithm for discovering clusters in large spatial databases with noise.,' in *kdd*, vol. 96, 1996, pp. 226–231.

[11] M. Ribeiro, A. E. Lazzaretti and H. S. Lopes, 'A study of deep convolutional auto-encoders for anomaly detection in videos,' *Pattern Recognition Letters*, vol. 105, pp. 13–22, 2018.

[12] A. Nanduri and L. Sherry, 'Anomaly detection in aircraft data using recurrent neural networks (rnn),' in *2016 Integrated Communications Navigation and Surveillance (ICNS)*, Ieee, 2016, pp. 5C2–1.

[13] D. Bank, N. Koenigstein and R. Giryes, 'Autoencoders,' *arXiv preprint arXiv:2003.05991*, 2020.

[14] B. Schölkopf, R. C. Williamson, A. Smola, J. Shawe-Taylor and J. Platt, 'Support vector method for novelty detection,' *Advances in neural information processing systems*, vol. 12, 1999.

[15] I. Goodfellow, J. Pouget-Abadie, M. Mirza *et al.*, 'Generative adversarial networks,' *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[16] S. Hochreiter and J. Schmidhuber, 'Long short-term memory,' *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[17] K. Greff, R. K. Srivastava, J. Koutnık, B. R. Steunebrink and J. Schmidhuber, 'Lstm: A search space odyssey,' *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2016.

[18] Y. Ji, L. Wang, W. Wu, H. Shao and Y. Feng, 'A method for lstm-based trajectory modeling and abnormal trajectory detection,' *IEEE Access*, vol. 8, pp. 104 063–104 073, 2020.

[19] A. Sherstinsky, 'Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network,' *Physica D: Nonlinear Phenomena*, vol. 404, p. 132 306, 2020.

[20] I. Kontopoulos, A. Makris, D. Zissis and K. Tserpes, 'A computer vision approach for trajectory classification,' in *2021 22nd IEEE International Conference on Mobile Data Management (MDM)*, IEEE, 2021, pp. 163–168.

[21] A. Bosch, A. Zisserman and X. Munoz, 'Image classification using random forests and ferns,' in *2007 IEEE 11th international conference on computer vision*, Ieee, 2007, pp. 1–8.

[22] J.-P. Jodoin, G.-A. Bilodeau and N. Saunier, 'Urban tracker: Multiple object tracking in urban mixed traffic,' in *IEEE Winter Conference on Applications of Computer Vision*, IEEE, 2014, pp. 885–892.

[23] U. Jaitley. 'Why data normalization is necessary for machine learning models.' (2018), [Online]. Available: https://medium.com/@urvashilluniya/why - data - normalization - is - necessary - for - machine - learning - models-681b65a05029.

[24] P. Baldi, 'Autoencoders, unsupervised learning, and deep architectures,' in *Proceedings of ICML workshop on unsupervised and transfer learning*, JMLR Workshop and Conference Proceedings, 2012, pp. 37–49.

[25] A. Paszke, S. Gross, F. Massa *et al.*, 'Pytorch: An imperative style, high-performance deep learning library,' in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[26] R. Sebastian, 'Model evaluation, model selection, and algorithm selection in machine learning,' in 2020, ch. 3.4.

[27] L. R. Olsen. 'Multiple-k: Picking the number of folds for cross-validation.' (2018), [Online]. Available: https://cran.r-project.org/web/packages/cvms/vignettes/picking_the_number_of_folds_for_cross-validation.html#:~:text=A%5C%20higher%5C%20k%5C%20(number%5C%20of,more%5C%20of%5C%20the%5C%20available%5C%20data..

[28] T. O. Hodson, T. M. Over and S. S. Foks, 'Mean squared error, deconstructed,' *Journal of Advances in Modeling Earth Systems*, vol. 13, no. 12, e2021MS002681, 2021.

[29] D. P. Kingma and J. Ba, 'Adam: A method for stochastic optimization,' *arXiv preprint arXiv:1412.6980*, 2014.

[30] S. Ruder, 'An overview of gradient descent optimization algorithms,' *arXiv preprint arXiv:1609.04747*, 2016.

[31] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy and P. T. P. Tang, 'On large-batch training for deep learning: Generalization gap and sharp minima,' *arXiv preprint arXiv:1609.04836*, 2016.

[32] A. F. Agarap, 'Deep learning using rectified linear units (relu),' *arXiv preprint arXiv:1803.08375*, 2018.

[33] Martín Abadi, Ashish Agarwal, Paul Barham *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[34] R. Mohammed, J. Rawashdeh and M. Abdullah, 'Machine learning with oversampling and undersampling techniques: Overview study and experimental results,' in *2020 11th international conference on information and communication systems (ICICS)*, IEEE, 2020, pp. 243–248.

[35] U. Jaitley. 'An overview of categorical input handling for neural networks.' (2019), [Online]. Available: https://towardsdatascience.com/an-overview-of-categorical-input-handling-for-neural-networks-c172ba552dee.

[36] J. T. Hancock and T. M. Khoshgoftaar, 'Survey on categorical data for neural networks,' *Journal of Big Data*, vol. 7, no. 1, pp. 1–41, 2020.

[37] K. Janocha and W. M. Czarnecki, 'On loss functions for deep neural networks in classification,' *arXiv preprint arXiv:1702.05659*, 2017.

[38] (), [Online]. Available: https://www.geogebra.org/.