Department of Computer Science

UNIVERSITY of York

Submitted in part fulfilment for the degree of
MSc in Cybersecurity.

Abnormal movement detection with trajectory and deep learning

Dragos Stoican

Version 2.20, 2023-January-23

Supervisor: Kofi Appiah

Contents

1 Introduction

Abnormal trajectory detection is a crucial task that can be utilized in various fields such as surveillance, transportation, or healthcare. The ability to identify abnormal patterns of movement can help prevent security threats, improve traffic management, or detect early signs of disease. All of this can lead to enhanced safety, efficiency, and well-being.

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 traditional machine learning methods for anomaly detection like isolation forest or clustering algorithms fail to interpret large amounts of multi-dimensional data, research is moving towards more powerful, more computationally expensive deep learning models. 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.

In this project, I propose and analyze various methods for abnormal trajectory detection in traffic surveillance scenarios, with the aim of presenting the most viable state-of-the-art solutions, tackling the challenge of working with big data, tailoring models to sequential data, evidencing the potency and efficacy of deep learning methodologies, comparing benefits of supervised and unsupervised learning, while keeping in mind the hurdle of real-world implementations of such systems.

2 Literature review

2.1 Deep Autoencoders and Generative Adversarial Networks

In 2018, Roy and Bilodeau propose a solution [1] for detecting abnormal trajectories in road intersections, using a neural network model called a deep autoencoder (DAE) [2]. 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 [3], Isolation forest [4], and vanilla autoencoders [2] and discover that it outperforms all in True Positive Rate and False Positive Rate for normal and abnormal trajectories.

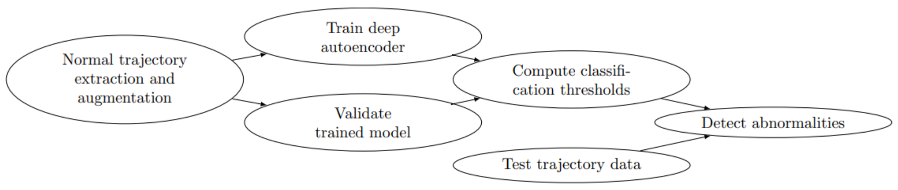

Figure 2.1: The main steps of abnormal trajectory detection using a DAE

One year later, Roy and Bilodeau [5] continue their research by improving their implementation in [1]. The DAE architecture is built upon by adding a discriminator inspired by Generative Adversarial Networks (GAN) [6]. 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 [1]. 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.

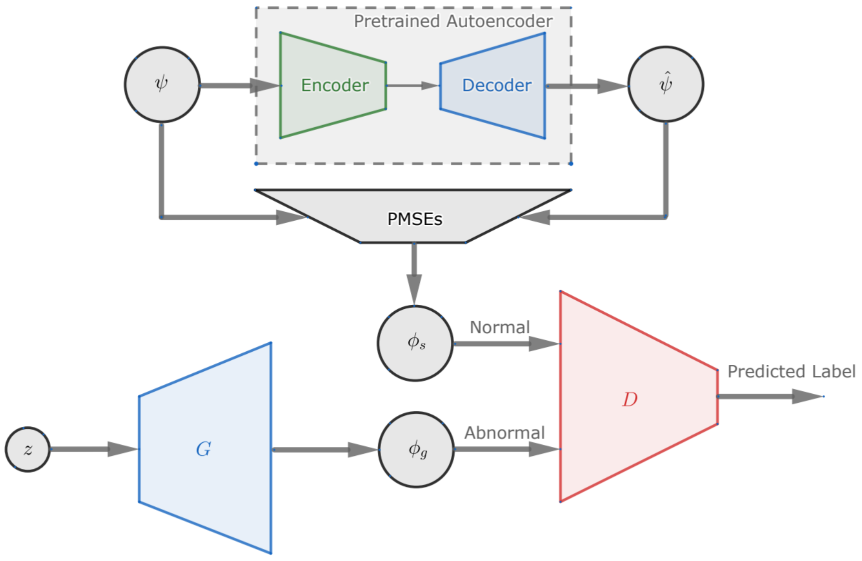

Figure 2.2: The normal trajectory input ψ and the reconstructed input $\hat{\psi}$ are used to calculate the reconstruction error (RE) ϕ_s using partial mean squared error (PMSE). The generator G creates REs ϕ_g that could be classified as abnormal, and the discriminator D learns to distinguish between the two in order to classify the data (Can't really cut this down, so I am thinking about removing it if it's too long or too irrelevant)

An AE is also used by M. Ribeiro et al. in [7], 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 [1], 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 [8], this structure can be exploited by other deep learning methods.

Gupta et. al. also propose a GAN for predicting future behaviors of pedestrians in [9]. 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

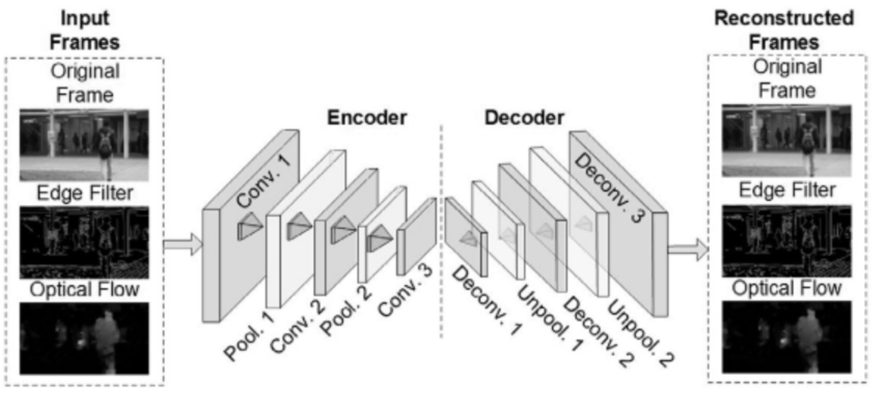

Figure 2.3: The aim of this network is to output reconstructed frames that are as similar as possible to the input frames. The hidden layers hold knowledge about how to interpret images, or how to recognize features in them

due to the addition of Long Short-Term Memory (LSTM) [10] [11] layers, which specialize in making use of long sequences of data, as shown in other papers discussed here [8], [12].

2.2 Long Short-Term Memory and Recurrent Neural Networks

Long Short-Term Memory (LSTM) networks, introduced by S. Hochreiter in [10], and further explored by K. Greff or Alex Sherstinsky in [11] and [13] 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 [8] 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.

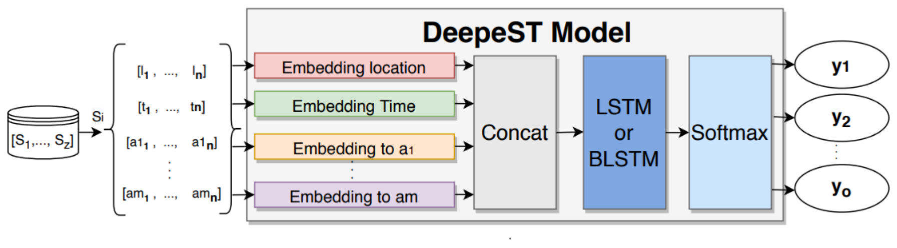

Figure 2.4: Subtrajectories S_i contain multiple sequential features, which are embedded and concatenated into a single array that is inputted into the LSTM layer, whose output can be used to classify the data

In [12] 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 [8]. Instead they use deep learning algorithms to obtain sequence-type trajectory data via RNNs, removing the need to extract 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 [14] proposes an LSTM-based neural network combined with the clustering algorithm DBSCAN [15]. 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

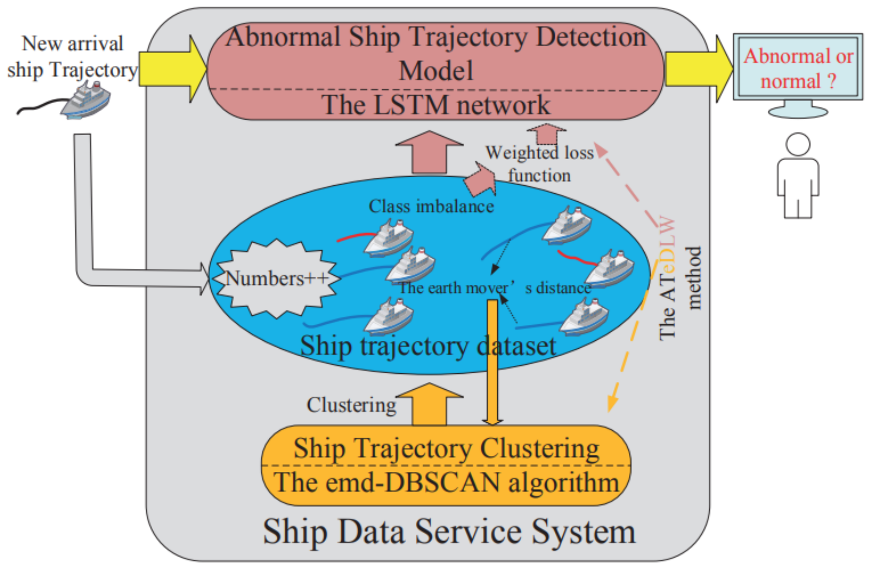

Figure 2.5: The DBSCAN algorithm clusters trajectories based on the earth mover's distance, and the velocities of boats. The clustered dataset is used to train the LSTM network, but due to class imbalance, a weighted loss function has to be used. New trajectories can then be classified by the trained model

network are the ones obtained by A Nanduri in [16]. 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 [8], he also demonstrates that older techniques suffer from the curse of dimensionality of the data, having to reduce the dimensions 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 [17]. 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 [18] 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 will be a reproduction of Roy and Bilodeau's deep autoencoder (DAE) network in [1], and it will serve as a benchmark for comparison of other methods. This implementation will also be used in an attempt to justify Roy and Bilodeau's choices of training hyperparameters like the optimizer or loss function. The second and third implementations will aim to exploit the sequential structure of trajectory data using recurrent neural networks, which is something that Roy and Bilodeau don't take full advantage of. The differences between supervised and unsupervised learning will also be covered and discussed with regard to ease of deployment in a real-world application.

3.1 Datasets

The datasets used are adopted from [1], 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 [19], 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 [19], 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 three sets of abnormal entries. The first set consists of straight lines with constant velocities that could be considered "clearly abnormal", while the other two include 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. Figures 3.1 and 3.2 show the three

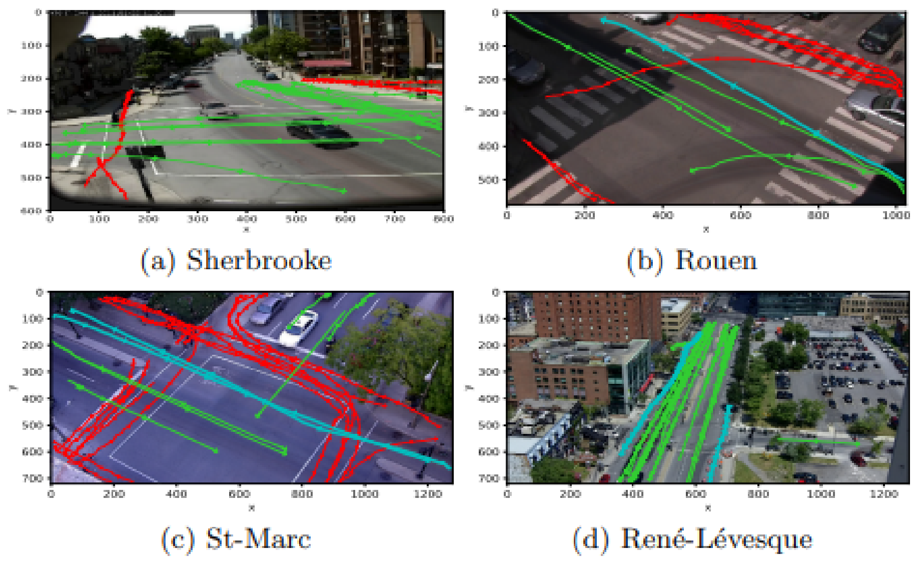

Figure 3.1: Original annotated trajectories of the Urban Tracker Dataset (image taken from [19]). Red, green, and blue represent pedestrians, cars, and bikes respectively. Note that Sherbrooke contains no bicycles and Rene contains no pedestrians. All trajectories are considered to be normal.

abnormal subsets of trajectories for the St-Marc intersection. Abnormal subsets for all the other datasets can be seen in Appendix A. These figures were generated using the ‘show_trajectories.py‘ script.

Training a deep neural network requires much more data than what is available in the Urban Tracker Dataset, which is why augmentation techniques had to be used by Roy and Bilodeau 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. 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.

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.

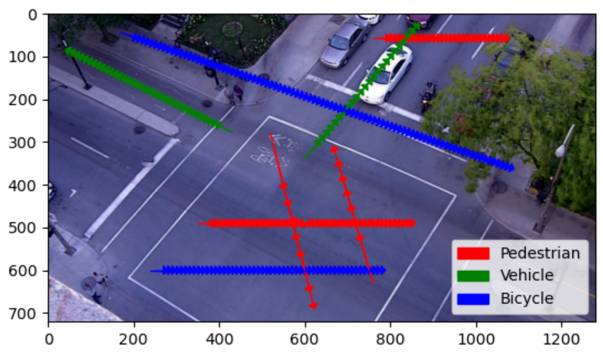

Figure 3.2: Simple abnormal data for the St-Marc dataset. Consists of jaywalking pedestrians crossing the road at inappropriate locations, cars on the cycle path or on the wrong side of the road, and bicycles cutting across the street or on the pavement

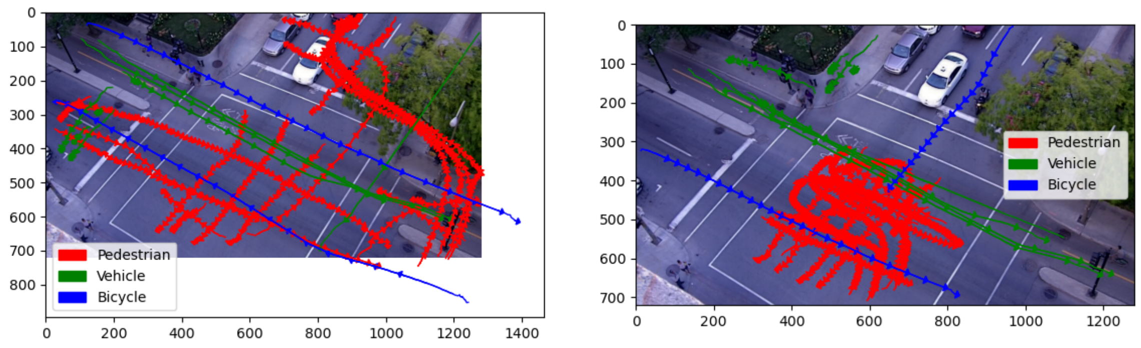

Figure 3.3: The realistic sets of abnormal trajectories, with pedestrians on the road, cars on the cycle path or on the pedestrian crossings, and bicycles on the pavement or on the road. These trajectories are more complex in velocities and are usually not only straight lines

While the data is now in the correct shape and contains all the entries needed for training and testing, Roy and Bilodeau's 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

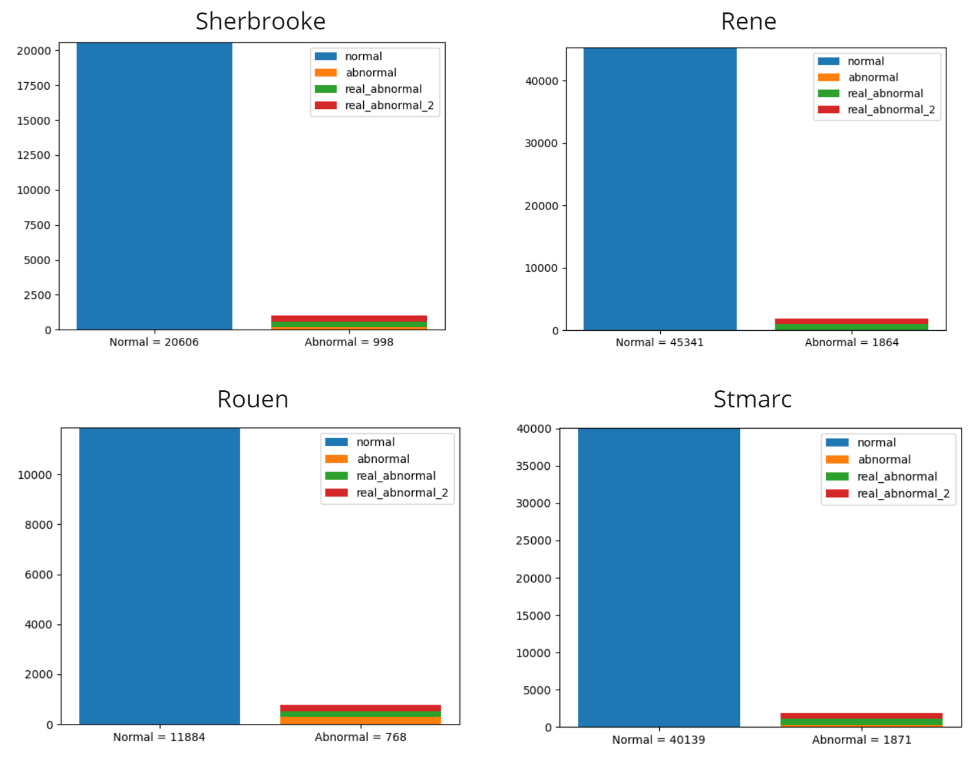

Figure 3.4: Data distribution between normal and abnormal trajectories for each dataset. Interesting to note the high class imbalance, but this is due to the fact that only normal data is used for training. Abnormal data is only necessary for testing and so it doesn't require the same size

fact, features in the trajectory data will have different ranges, like world coordinates and velocity, which are measured on different scales. Also, this helps the network learn useful features faster and negates the effect of outliers while preserving the shape of the original distribution [20]. Appendix B includes experiments conducted with and without normalization on the Sherbrooke dataset and the benefits of its addition to the data preprocessing pipeline is clear. Data normalization was implemented in the data_preprocessing.py script.

3.2 Deep autoencoder

3.2.1 Autoencoders

An autoencoder is a type of fully-connected artificial neural network that is used for unsupervised learning [2]. 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 works like any other autoencoder, but it consists of a lot more layers, making it better suited for working with big data.

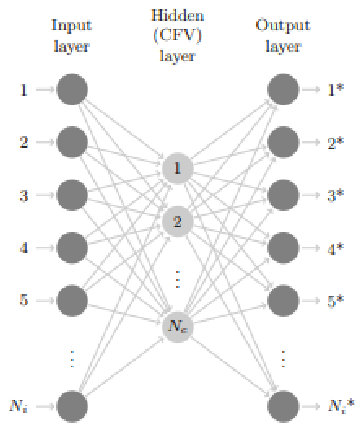

Figure 3.5: Diagram showing the simplest autoencoder.

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 DAE_method.py script.

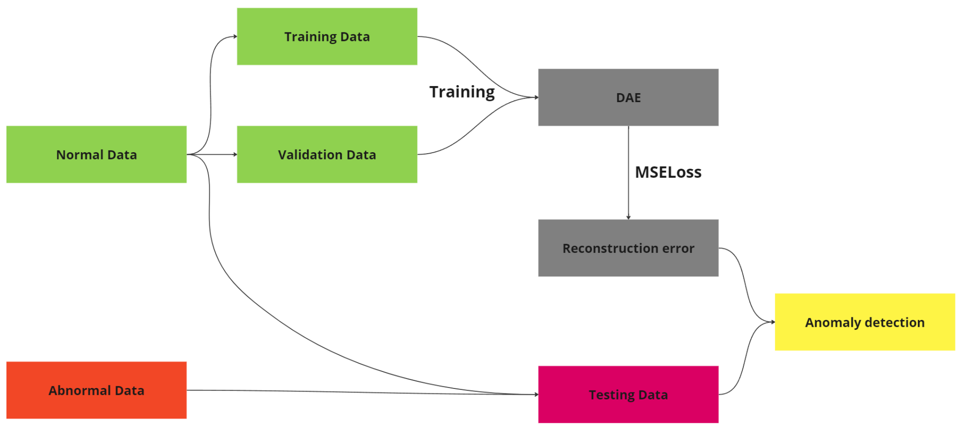

Figure 3.6: Diagram showing the simplest autoencoder.

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. 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 [21]. Appendix C 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.

The loss function used is Mean Squared Error (MSE), computed between the output of the model and the original trajectory. Roy and Bilodeau justifiably chose this loss function in [1], which is commonly used in sequence-to-sequence networks [16]. Another option here would be Binary Cross Entropy Loss, as seen in [14], 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 D 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 [1], as I have opted to go for the more traditional Adam optimizer, as opposed to RMSProp. The decision to use RMSProp is not justified by Roy and Bilodeau, and Adam has consistently provided better results in my tests (see Appendix E). In fact, this optimizer is more utilized across similar literature, with [9] [16] 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 [22]. A condensed representation of all training hyperparameters can be seen in Table 3.1

Table 3.1: DAE training parameters

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

3.2.4 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. 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 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 [7]. While his implementation used convolutional layers instead of linear layers, the similarities between the encoder and decoder functionalities are obvious.

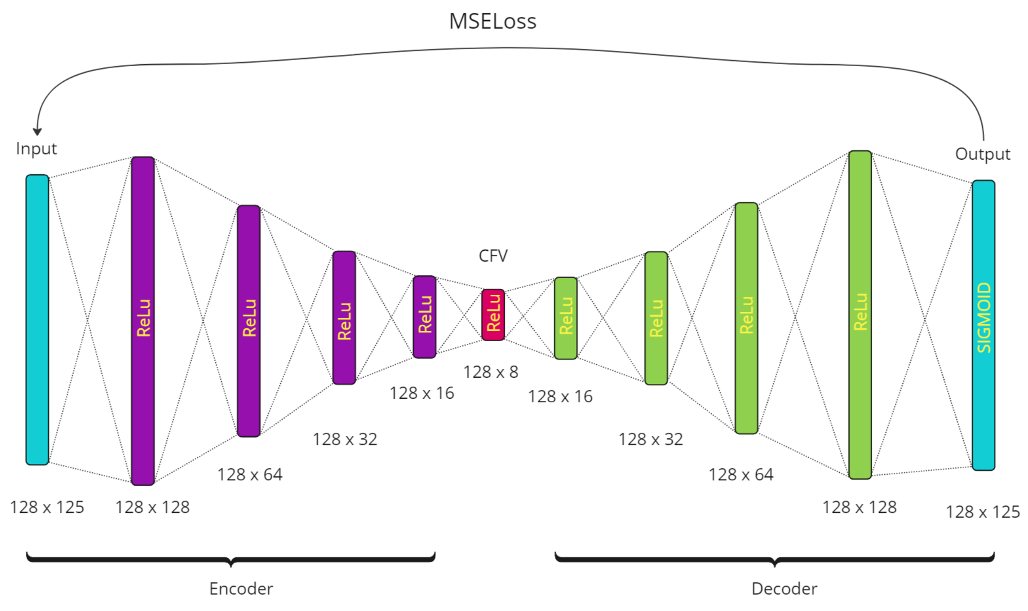

Figure 3.7: Diagram showing the DAE architecture, where the most interesting thing to note is the input and the output of the network, which have to correspond in dimensions after the data has been compressed and decompressed while passing through the hidden layers. The process goes from left to right, 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 Recurrent Neural Network

3.3.1 Long-Short-Term-Memory cells

LSTM networks, composed of LSTM cells, are a type of RNN that was designed to fix the problem of vanishing gradients [13] [10]. 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.

It is clear why this architecture could prove very efficient in dealing with trajectory data, as Freitas showed in [8], 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 input and output a single value, between 0 and 1, which represents the confidence that the given trajectory was abnormal. 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.

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

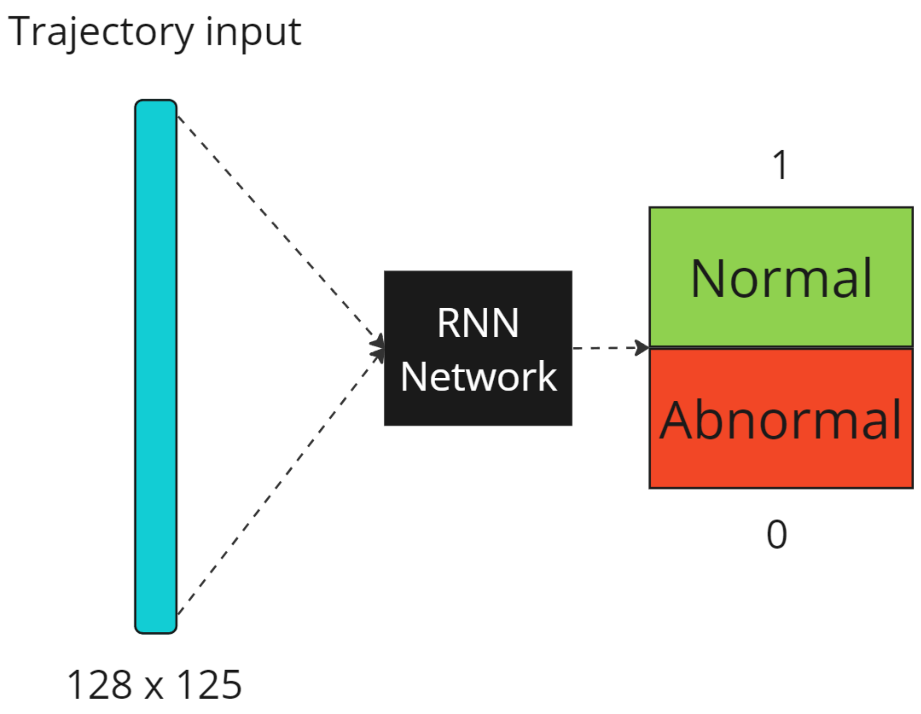

Figure 3.8: Trajectory data is passed into the network, which outputs a value between 0 and 1. A value greater than 0.5 means that the trajectory was normal, while a value lower than 0.5 means that the trajectory is abnormal.

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 [23]. 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 3.7 shows the results of the class balancing technique on the Sherbrooke dataset.

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 [24]. There has been a lot of

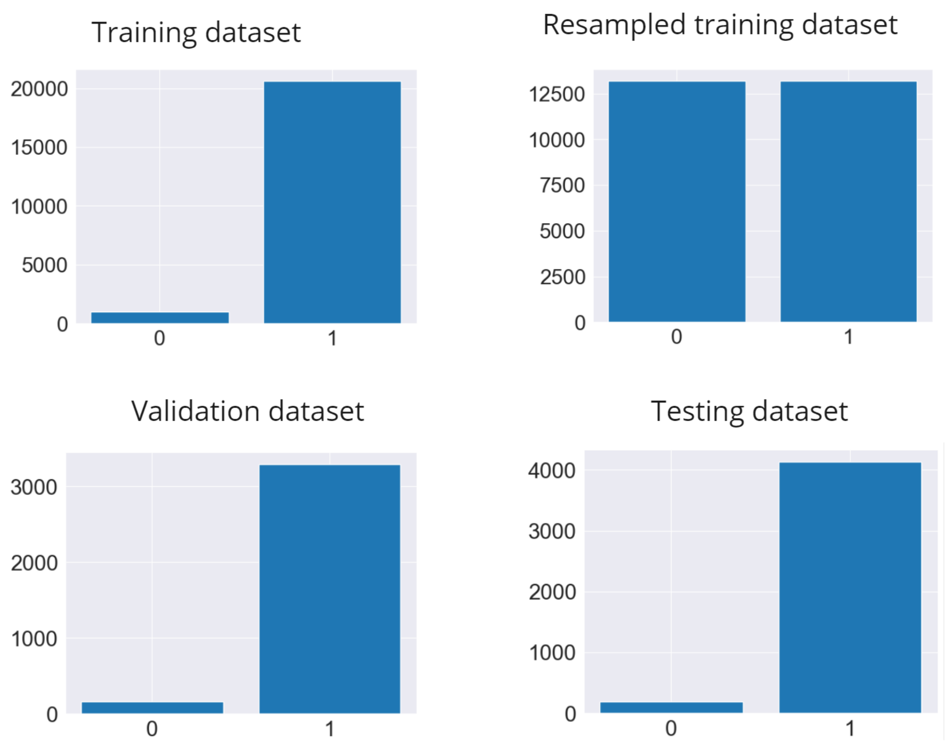

Figure 3.9: After oversampling, the training dataset is balanced (equal number of normal and abnormal entires). However, testing and validation datasets are kept unbalanced, since those are used to evaluate the performance of the network.

research on the subject of encoding categorical features [25], 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 3.8. It is interesting to mention the similarities to the input format in Figure 2.4, where Freitas [8] aims to make use of the sequential nature of trajectory features, demonstrating the benefits of large multi-featured sequential data.

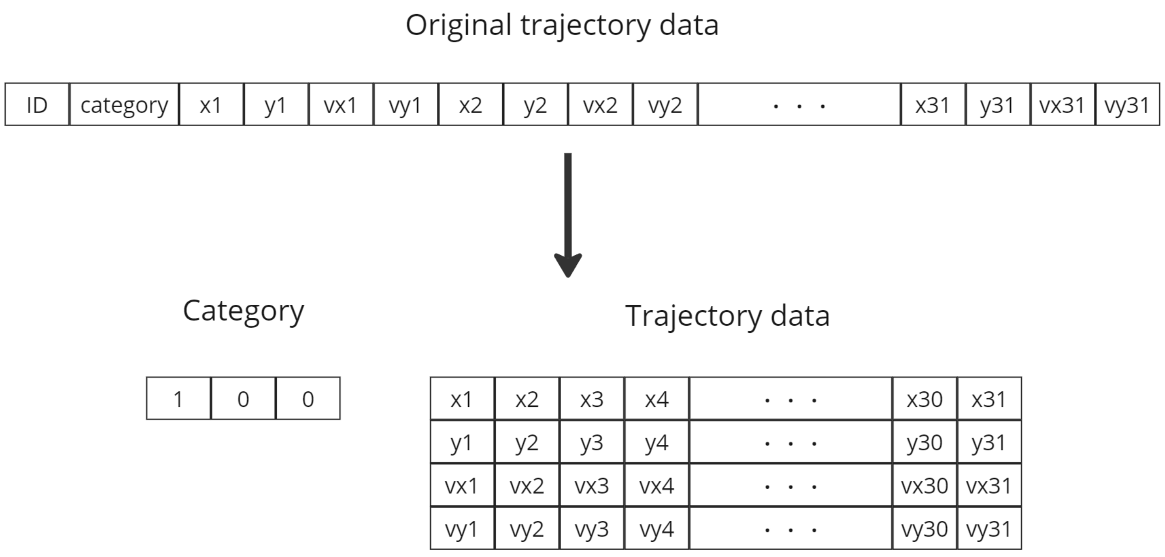
Original trajectory data
ID | category | x1 | y1 | vx1 | vy1 | x2 | y2 | vx2 | vy2 | · · · | x31 | y31 | vx31 | vy31
Category
1 | 0 | 0
Trajectory data
x1 | x2 | x3 | x4 | · · · | x30 | x31
y1 | y2 | y3 | y4 | · · · | y30 | y31
vx1 | vx2 | vx3 | vx4 | · · · | vx30 | vx31
vy1 | vy2 | vy3 | vy4 | · · · | vy30 | vy31

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 RNN architecture

The 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 3.9 shows a diagram representing this architecture.

MENTION SIMILARITIES TO FIGURE 2.4 HERE SOMEHOW!! THE MULTI FEATURES!!

The loss function used in training is Binary Cross Entropy (BCEL) since it widely considered the best option for binary classification problems, and

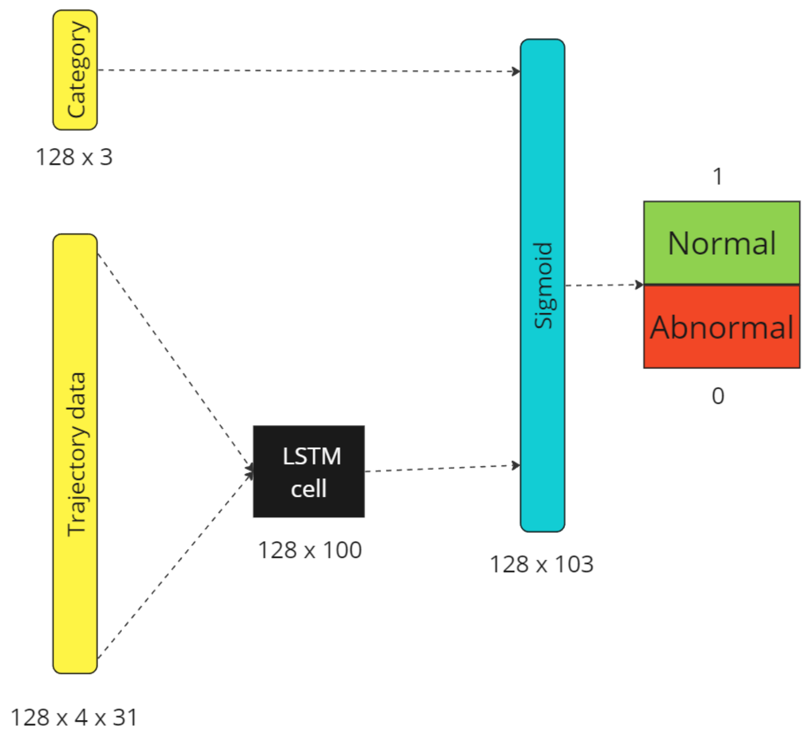

Figure 3.11: The novelty of this architecture lies in the multiple-input model. One input is the trajectory data which is passed to the LSTM cells, while the other input is the categorical data, which doesn't follow a sequential logic and so it skips the LSTM layer. The output of the LSTM layer and the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

unlike the deep autoencoder method we are not dealing with sequence-to-sequence networks anymore, where MSE was the better choice, but a sequence-to-one network. The optimiser is once again Adam with a learning rate of 0.001, as this is the most common choice among similar papers, like [3] and (add link). Training takes place over only 20 epochs, as the model quickly converges to a very good solution.

Table 3.2: RNN training parameters

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

4 Results and analysis

4.1 Testing methodology and metrics

The results are generated by testing the two models on the four datasets presented in section 3.1, after creating the networks as 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 be focusing 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 an extremely valuable metric that I will be using to analyze predictions. The confusion matrix is a 2x2 matrix composed of 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.

4.2 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 on 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 E contains similar results for the other three datasets.

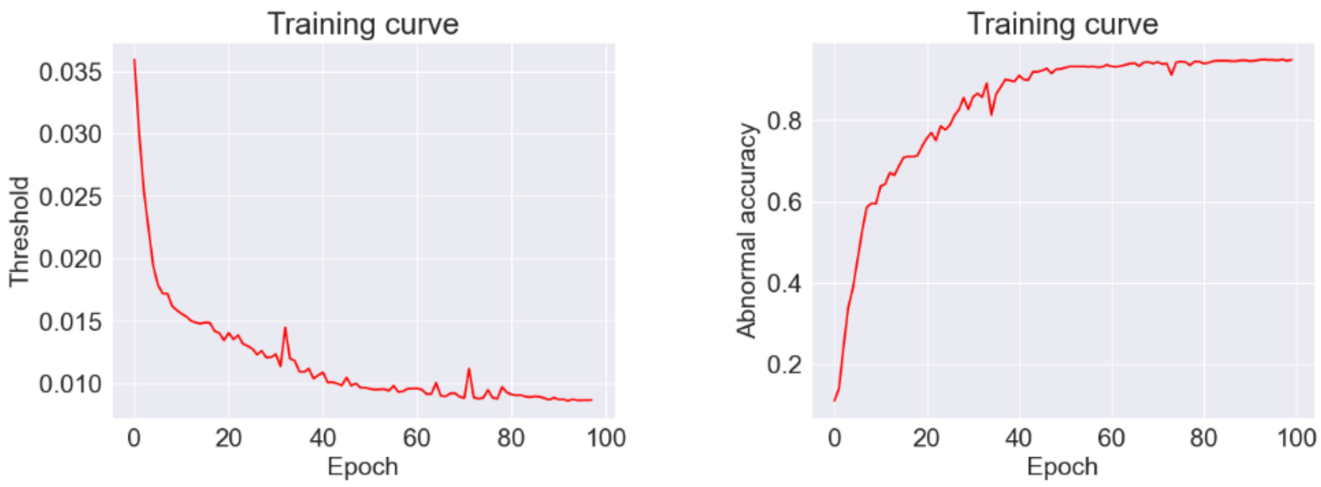

Figure 4.1: On the left, the evolution of the threshold value across the 100 iterations, on the right the accuracy obtained by the model on the abnormal data at every one of those iterations. The correlation between the two is clear, even in the spikes around epoch 35 and epoch 75.

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. These results follow the same structure as the results obtained by Roy and Bilodeu in [1], therefore the model is only tested on realistic abnormal data. A comparison to the results in [1] can be seen in Table 4.1, and the difference in accuracy can be attributed to training epochs, the change in

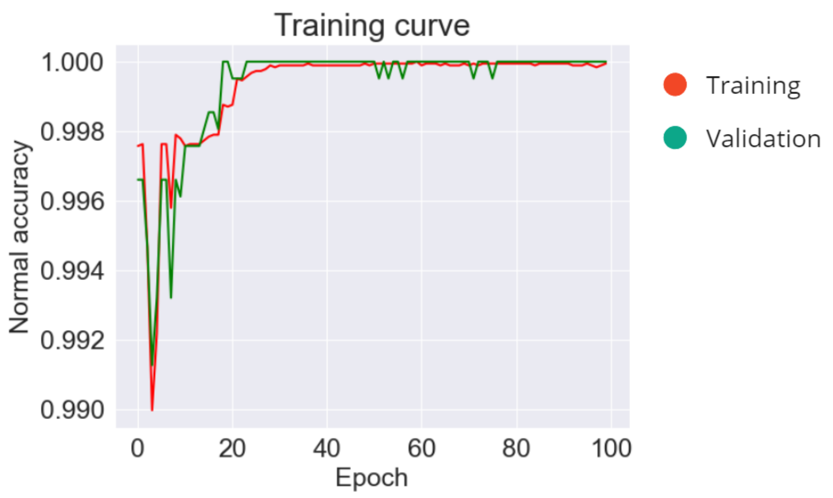

Figure 4.2: This plot shows that the accuracy of the model on the normal data never drops under 99%. In fact, the lowest accuracy is obtained at the start of the training process, but after a few iterations, the model maximizes its accuracy on the training data while still reducing the threshold value.

optimizer, or other hyperparameter differences that are not clearly described in Roy and Bilodeau's implementation.

Table 4.1: DAE results

model	sherbrooke	rene	rouen	stmarc
Original DAE	80	85	32	39
My DAE	75	87	28	12

4.3 RNN 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 higher amount of manual intervention for labeling the dataset.

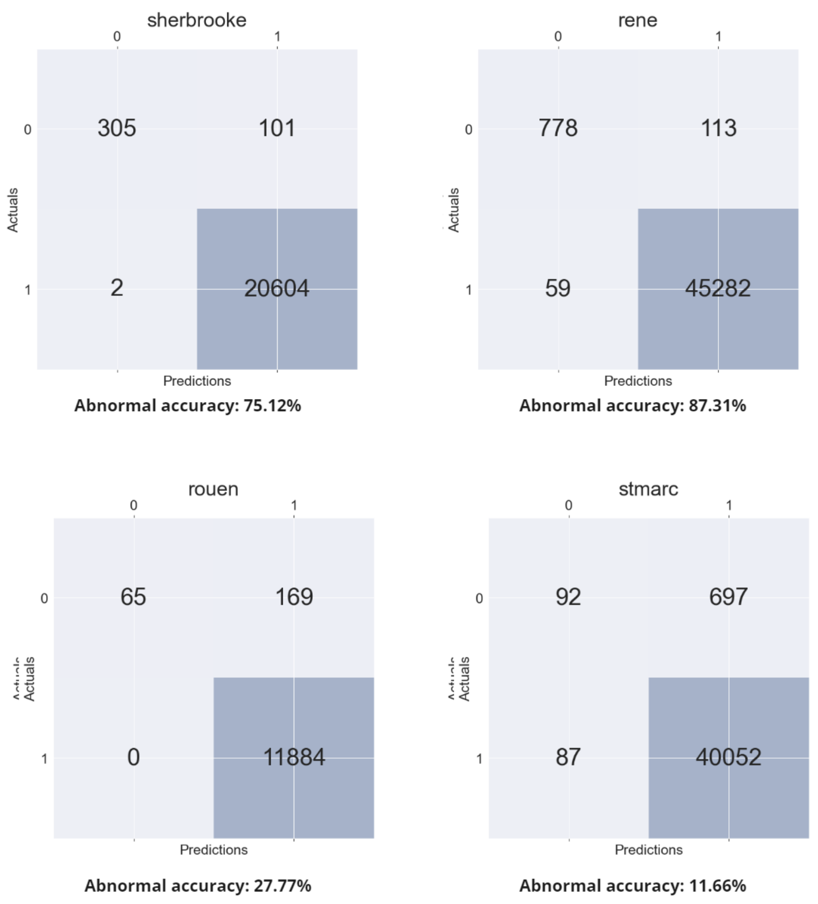

Figure 4.3: DAE method results

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 trajecctories that the model doesn't see during training. This subset is smaller than the set of trajectories that was used in testing the DAE, but that doesn't invalidate the results.

Looking at the training metrics here, it is clear that the network is having a really 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 increase in performance until the model eventually converges to a final solution.

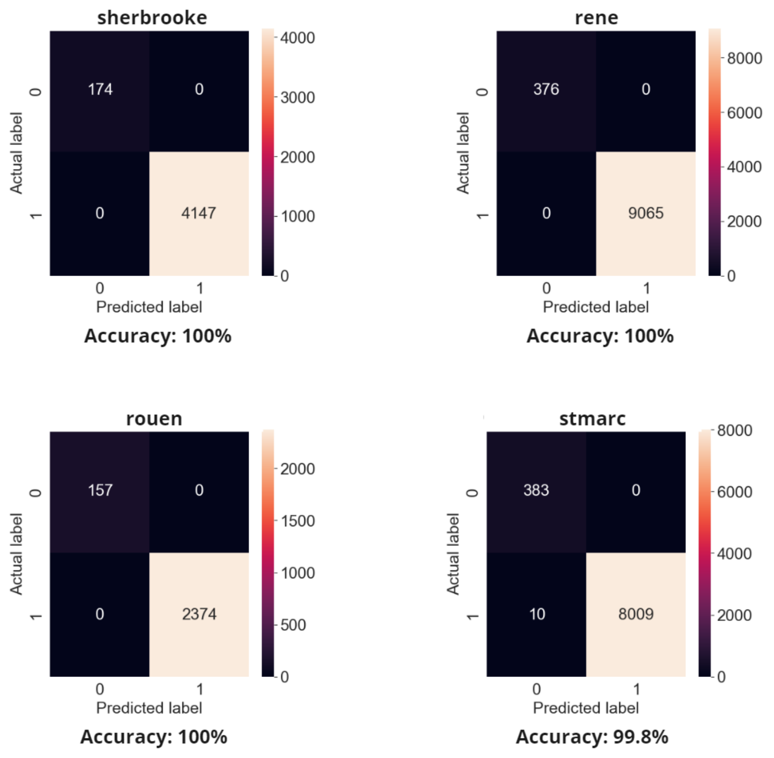

Figure 4.4: The RNN achieves almost perfect accuracy on all the datasets, the only mistakes are done on normal data on the St-Marc dataset, but abnormal trajectories are always classified correctly.

4.3.1 Clustering unlabelled data

4.4 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 autoencoder to perform as well as the RNN, and what type of entries it struggles with the most

4.4.1 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.

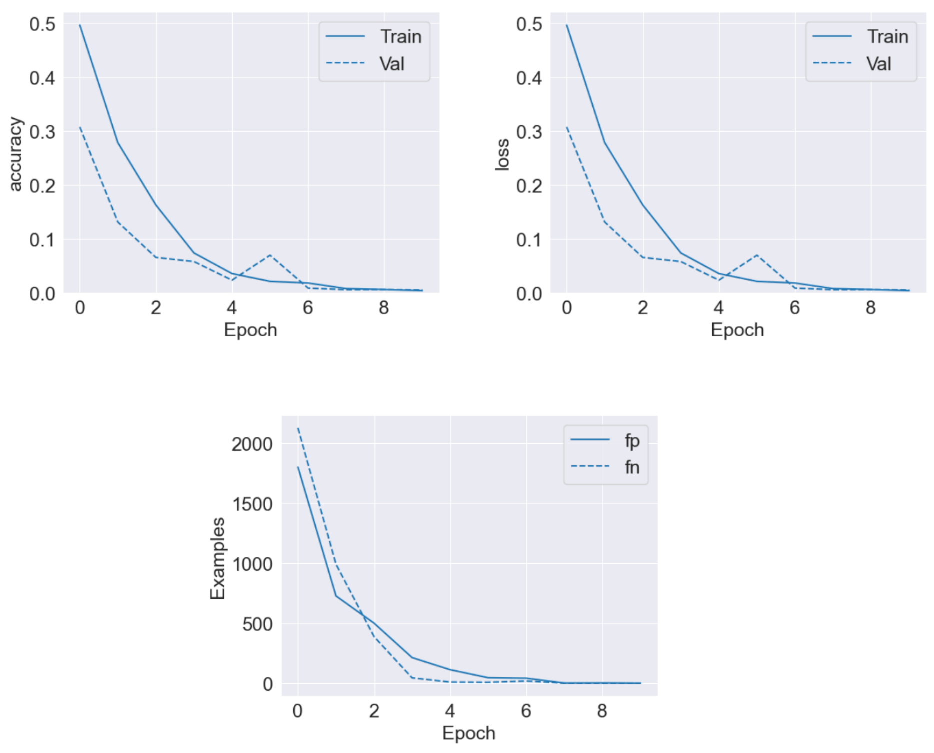

Figure 4.5: RNN method results

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. The original entries are created using the GeoGebra platform [26]. Figure 4.6 shows the original data in the york dataset.

4.4.2 York dataset results

Comparing the two models head to head is still not perfectly clear as the supervised learning approach only used a subset of the dataset for training, while the DAE can use the entire data for testing. However, it is clear from the results in Figure 4.7 that the DAE method is having a hard time classifying the abnormal trajectories in this dataset, while the RNN method is once again only misclassifying a handful of normal trajectories. What is interesting to note however is the perfect accuracy of the DAE on the normal data, which is not achieved in any of the other four datasets and is

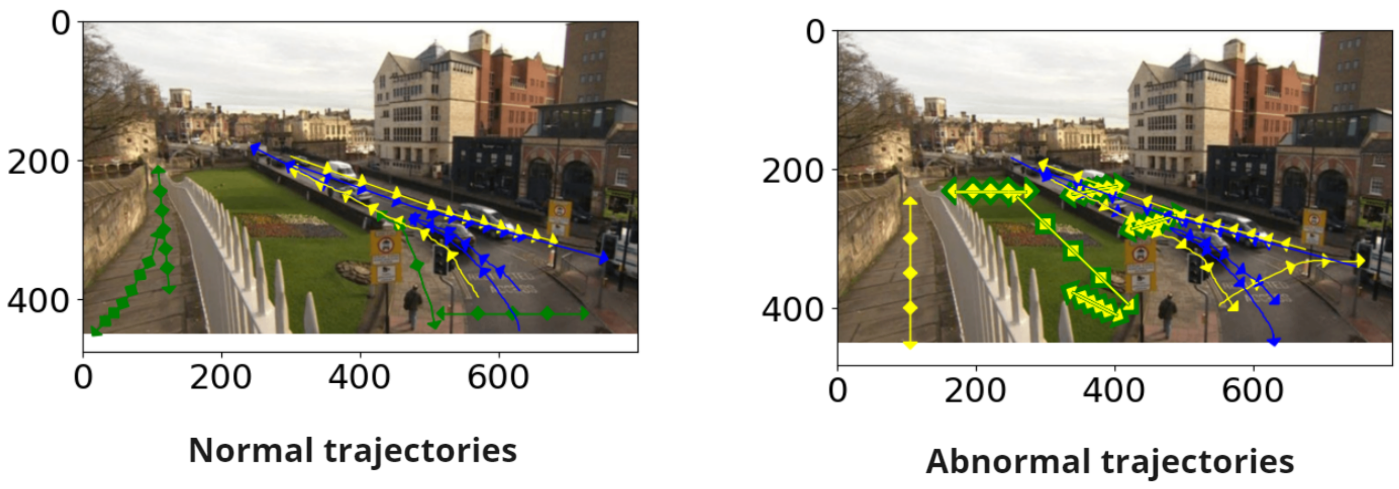

Figure 4.6: York dataset. Green for pedestrians, blue for vehicles, and yellow for bicycles

a better performance than the RNN could achieve. This could suggest that in an environment where true negatives are a serious concern, the DAE method would be preferred.

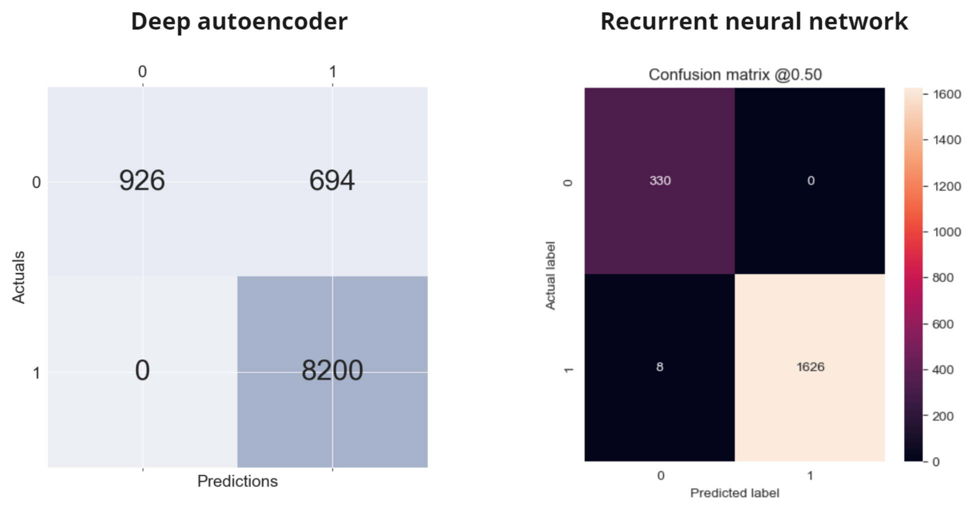

Figure 4.7: Result comparison

4.5 Final analysis and comparison of the two approaches

Ultimately the RNN method performs significantly better on all datasets, but the decision between using one or the other in a real-world situation can depend on other factors, not only the performance of the network. Most notably, the RNN network uses supervised learning to train on a large amount of data. In a real-world situation, this would require a very large amount of labeling of data, and the problem of lack of abnormal data would be a massive issue as the datasets get larger.

On the other hand, while this architecture of the autoencoder doesn't perform as well as the RNN, it has the big advantage of not needing abnormal data for training and that the data doesn't have to be labeled for training. This represents a huge benefit in the rapid deployment of such a system since it would be capable of learning normal trajectories in any scene after only collecting a relatively small amount of data, which doesn't have to be reviewed.

5 Conclusion and future work

In conclusion, the problem of detecting abnormal trajectories can be solved in a multitude of ways. Approaches using autoencoders, generative adversarial networks, recurrent neural networks, image classification, and data clustering have all been explored throughout time, as this challenge has been thoroughly explored. The two methods for solving abnormal trajectory detection that I present both work well in doing so, especially the Recurrent Neural Network. However, in a real-world application, this network has a set of drawbacks that would make it harder to implement, while the autoencoder takes advantage of unsupervised learning to require as little human intervention as possible.

Bibliography

[1] G.-A. B. Pankaj Raj Roy, ‘Road user abnormal trajectory detection using a deep autoencoder,’ 2018.

[2] D. B. et al., ‘Autoencoders,’ 2021.

[3] B. S. et al., ‘Support vector method for novelty detection,’ 1999.

[4] K. M. T. Fei Tony Liu, ‘Isolation forest,’ 2009.

[5] G.-A. B. Pankaj Raj Roy, ‘Adversarially learned abnormal trajectory classifier,’ 2019.

[6] I. J. G. et al., ‘Generative adversarial networks,’ 2014.

[7] M. R. et al., ‘A study of deep convolutional auto-encoders for anomaly detection in videos,’

[8] N. C. A. de Freitas et al., ‘Using deep learning for trajectory classification,’

[9] A. G. et al., ‘Social gan: Socially acceptable trajectories with generative adversarial networks,’ 2018.

[10] S. H. et al., ‘Long short-term memory,’ 1997.

[11] K. G. et al., ‘Lstm: A search space odyssey,’ 2017.

[12] Y. J. et al., ‘A method for lstm-based trajectory modeling and abnormal trajectory detection,’

[13] A. Sherstinsky, ‘Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network,’ 2021.

[14] T. Z. et al., ‘Atedlw: Intelligent detection of abnormal trajectory in ship data service system,’ 2021.

[15] M. E. et al., ‘A density-based algorithm for discovering clusters in large spatial databases with noise,’ 1996.

[16] L. S. Anvardh Nanduri, ‘Anomaly detection in aircraft data using recurrent neural networks (rnn),’

[17] I. K. et al., ‘A computer vision approach for trajectory classification,’ 2021.

[18] A. B. et al., ‘Image classification using random forests and ferns,’ 2007.

[19] J.-P. J. et al., 'Urban tracker: Multiple object tracking in urban mixed traffic,' 2013.

[20] 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.

[21] 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%20higher%20k%20(number%20of,more%20of%20the%20available%20data..

[22] N. S. K. et al., 'On large-batch training for deep learning: Generalization gap and sharp minima,' 2017.

[23] R. M. et al., 'Machine learning with oversampling and ndersampling techniques: Overview study and experimental results,' 2020.

[24] 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.

[25] J. T. Hancock* and T. M. Khoshgoftaar, 'Survey on categorical data for neural networks,' 2020.

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