Deep Unsupervised Representation Learning For Animal Activity Recognition

Rosalie Voorend

Supervisor: Dr. J.W. Kamminga
Critical observer: Dr. V.D. Le

Creative Technology BSc Thesis
University of Twente
Enschede, Netherlands
July 16 2021

Abstract

A generative model as unsupervised representation learning method is indicated to improve the performance of animal activity recognition (AAR), with intertial measurement unit (IMU) data. Since autoencoders performed well, the introduction of a variational autoencoder (VAE) to the AAR pipeline is tested and investigated. To do this, three feature extraction methods were built and compared. These were statistical feature extraction, unsupervised representation learning with a VAE and no feature extraction. Additionally, the size and type of the input data were altered to investigate the effects.

The results of this research showed that statistical feature extraction performed better than representation learning and no feature extraction. This increased the F-score of the AAR pipeline with 2%. Representation learning did not improve over using no feature extraction method; it scored an F-score that was 3% lower than the pipeline without feature extraction. It was concluded that the addition of feature extraction did improve the classification process, but the classifier that is used might not be compatible with the latent representations for unsupervised representation learning.

Contents

1 Introduction

Activity recognition is an area of machine learning and can be used to recognise actions from observations. Like a smart watch can recognize a workout, or how a smart alarm recognizes lighter sleeping behaviours. Activity recognition can be done in many ways and can provide insightful knowledge about any given data. For example, human activity recognition (HAR) can be used in a smart home to recognize what a user is doing and devices could react accordingly. In this case, animal activity recognition (AAR) will be explored. The benefits of recognizing activities of animals, is that the behaviour of animals could be analyzed. Domestically, this could help an owner monitor their pet whilst they are not at home. Or on a greater scale, AAR could greatly improve the well-being of wildlife, as well as the knowledge about ecosystems.

AAR is done with a machine learning algorithm that recognizes activities from raw data. One way to execute AAR, is to introduce a feature extraction method. In feature extraction, parts of the data are taken that represent the entire dataset. This can reduce the dimensionality of the data, for example. These features can be extracted manually, by computations on the data, or by a letting a neural network learn the representations of the data. The latter is called representation learning.

Feature extraction can be done in supervised and unsupervised manner. In supervised learning the selected raw data is organised and classified manually, which means it has to be reviewed an annotated by human labour. In unsupervised manner, this manual data annotation is not needed, since unlabeled data can be used. This unsupervised manner is representation learning. To increase speed and decrease human labour, it is important to explore ways of implementing unsupervised representation learning in the AAR process. This can be done using machine learning models.

Besides the tedious annotating labour, it is much easier to collect unlabelled data. The bottleneck of machine learning, and for example activity recognition, is not in the creativity of projects to apply it to, but in the availability of labeled data. Machines need labeled data to learn from. But since the labeling is so intensive, it is very insightful to look into the possibilities of using unlabeled data. Unsupervised representation learning is one of these possibilities. This unsupervised representation learning process could not only be applied to the AAR of this project, but it could potentially be adapted to other activity recognition projects.

Various feature extraction methods have been tested and evaluated for HAR and AAR. However, autoencoders and variational autoencoders (VAE) have barely been applied to raw inertial measurement unit (IMU) data for AAR. Especially not for the horse dataset that is used for this research. Therefore, the objective of this paper is to investigate an unsupervised representation learning method that results in the highest classification performance, from raw IMU animal activity data when using a state of the art VAE and comparing it to other feature extraction methods. This formulates the following research question: "To what extent does a variational autoencoder result in a better classification for animal activity recognition of raw IMU animal activity data?"

This research includes literature research on different methods of AAR and representation learning method for raw IMU data. Autoencoders as well as VAEs are explored for existing methods of HAR and AAR. Then, an approach to test a VAE and other feature extraction methods is described. Next, a VAE is built, trained and tested. This is followed by the creation of an AAR pipeline. This pipeline is then tested in three manners. Once using statistical feature extracion, then by using unsupervised representation learning with the VAE and lastly without any feature extraction. These variations of the pipeline are evaluated by using a vanilla classifier, that is built to fit the raw IMU

animal activity data. The results are discussed and and compared with regards to the IMU animal activity data. Finally, a conclusion is drawn based on the discoveries and gained insights.

2 Background

Feature extraction and representation learning for AAR can be done in many different ways. Some background information on the methods that are used in this project is described and explained below.

2.1 Animal And Human Activity Recognition

Firstly, the overarching goal of this project is AAR, since the dataset that is used consists of raw animal activity data from an IMU sensor. To make sense of this raw data, an AAR pipeline is created. AAR is very similar to HAR. However, HAR is more common and thus HAR is used for state of the art research. AAR, as well as its relations to HAR, is explained below.

AAR is the recognition of actions from animals. Often, this involves learning from observations of the activities from animals, using machine learning. This can be done by directly learning from the actions' observations or by first extracting features from the observation. In the case of the latter, the relation between features of the data is learned and with this a classifier can extract the activities from the raw data. AAR is very similar to HAR. The main difference is in the input data. HAR has the same process as AAR, but instead of having animal activity data as an input, HAR uses human activity data. HAR will therefore also result in a moderately different outcome than AAR, when using the same feature extraction and classification methods. Also, the applications of HAR and AAR differ. For example, HAR can be used in a smart watch or smart phone, to detect activities throughout the day. This can then immediately be used by the user. AAR on the other hand, is more for gaining insights, since animals likely do not use the data themselves. HAR and AAR are similar, in the sense that human activity data is the same type of data as animal activity data. In the scope of this project, they are both time-series sensor data. This is why the same process can be applied to both of them.

Nevertheless, there are still several distinctions between human and animal activity data. Firstly, there are slight differences in the movements of humans and animals in the real world that are also reflected in the datasets. The dataset for this research is horse activity data. The raw data that contains the activity *running*, will look different for horses, than for humans. This is depicted in Figure 1. Secondly, human activity data is commonly retrieved through a smart watch, which is worn around the wrist. Animal activity data is fetched through sensors which can be placed anywhere on the subject. The horse activity data of this project is collected through sensors worn around the neck. This influences the activity data. Another difference is that, generally speaking, horses also have a lower heart rate than humans [1]. This could interfere with the data. Another difference between raw animal data and raw human data, is that it is harder to place sensors on animals than on humans. Humans often voluntarily wear a smart watch to get insight on their data. A horse on the other hand, might find it annoying to wear a sensor and could try to get it off, which could cause deviations in the data. Lastly, the different sampling frequencies of different sensor might also influence results.

Figure 1: Raw IMU data of the activity *running* from a) a horse and b) a human

2.2 Feature Extraction

Next, the project uses three feature extraction methods: statistical feature extraction, representation learning and no feature extraction. To deeper understand what this means within the scope of the project, the basic concepts of each of this method is described below. Often, this means the raw data is reduced in dimensions, to make it more manageable for processing [2].

Feature extraction is the area of machine learning where parts of the original data are taken, which are representative for the entire data. It takes variables from the raw data and combines it into features, otherwise referred to as representations. An intuitive way to explain this, is the example using the following sentence: "The clawine flew to the tree, into its nest". "Clawine" is not an existing word in the English language, but from the context one could tell that a clawine is most likely a bird. There are words in the sentence that give it away, like "flew" and "nest". Likewise, when learning representations, there are priors that give away information about the data. This information is not definite, because clawine could also be an insect, but it is likely. There are several clues in the sentence that the representations are based on. It is important that representations describe the original dataset accurately. The preciseness of the representations has a big impact on how well a machine learning model works. Good representations make it easier to classify the raw data into the right categories. According to Bengio et al. [3], a good representation is "one that captures the posterior distribution of the underlying explanatory factors for the observed input". So this assumes there are underlying clues in the data, which a good representation can learn and it can also learn the impact of that underlying clue. Additionally, according to Bengio et al., a good representation is one that is useful and adds valuable information.

2.2.1 Statistical Feature Extraction

One form feature extraction is statistical feature extraction. The features that are taken from the data are statistical measures. For example, the mean can be taken from the data. The mean for *running* data, might be higher than the mean from *standing* data. This way, the mean could represent the data accurately. Not only the mean, but all statistical measures could be extracted from the data this way. These measures could be calculated for the entire dataset, or repeatedly for segments of the data.

2.2.2 Representation Learning

Another form of feature extraction is representation learning. Instead of extracting the representations directly, a neural network can be used to learn the representations. The network is trained to learn the representations and then the model of that network can be used to extract features. Representation learning can be done in semi-supervised or in unsupervised manner. Semi-supervised uses few labels to learn the representation from and unsupervised uses no labels at all. Using unsupervised representation learning, recognition of objects, images or speech could be automated, amongst others.

There are many ways to carry out unsupervised representation learning [4][5][6][7][8][9]. Deep learning methods that will be mentioned in the state of the art are Convolutional Neural Networks (CNN), Long Term Short Term Memory (LSTM), Recurrent Neural Networks (RNN), autoencoders and VAEs. The latter, VAEs, will be used within this project and is explained below.

2.3 Autoencoders And Variational Autoencoders

For this project, a VAE will be utilized as representation learning method. In order to give background information on VAEs, normal autoencoders should be explained first. Then, the basics of a VAE network are explained.

2.3.1 Autoencoders

Most machine learning networks consist of one network, which takes input data and produces an output. However, in an autoencoder, there are two of these networks, which are connected through a bottleneck. One is the encoder, the other the decoder. The encoder network and the decoder network, are put in the opposite fashion, as can be seen on the left of Figure 2. Generally speaking, the encoder part tries to compress the input data into a latent space, the middle of the left in Figure 2, and the decoder tries to reconstruct the same input data, from that latent space. Therefore, ideally, the input is the same as the output. As this latent space is a smaller layer than the encoder and decoder parts, it sort of forces the network to become a compressed version of the input data, which is why it could be seen as the bottleneck. Every neuron in the bottleneck is a combination of every input neuron, with varying weights. This also means that if the input features are independent of each other, learning a representation through this combination would be unfeasible.

The compression in the bottleneck is mandatory, because if there was no bottleneck, like on the right side of Figure 2, the autoencoder could just remember the input data and copy it for the output data, without learning the representations. The latter is called overfitting. The key is to find the balance between a good representation of the input data, without actually copying it.

One way to create a good bottleneck would be to lessen the number of neurons in the hidden layer(s). For this, the number of neurons in the bottleneck should be small enough so the autoencoder does not copy the input data. This is especially important to keep in mind for an autoencoder that uses deep networks. Here, a bottleneck with only one neuron could still be capable of memorizing the input data. With the right bottleneck, an autoencoder can learn by constantly looking at the reconstruction error.

Autoencoders are qualified to learn nonlinear relations, which makes it different from other methods of representation learning. Since they learn from trying to find relations, the input data does not need to be labelled. This makes autoencoders very well fitted as an unsupervised learning technique.

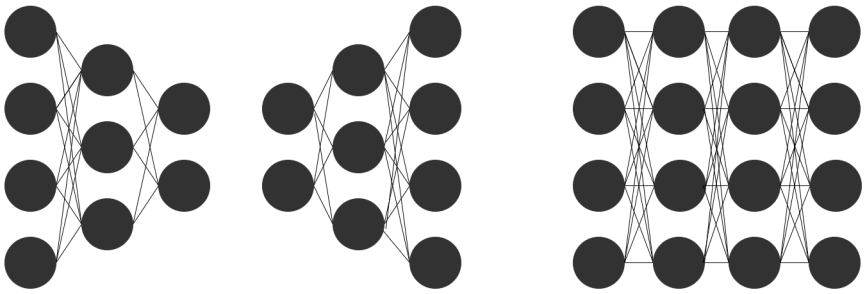

Figure 2: **Left**: A basic autoencoder architecture and **Right**: An autoencoder architecture without a bottleneck

2.3.2 Types Of Autoencoders

There are multiple types of autoencoders, that are slight variations on a vanilla autoencoder. Two of these are mentioned in the state of the art, hence they are briefly explained below.

Sparse autoencoders are a type of autoencoders where only some of the neurons are active for each observation in the data. Which neuron is active depends on the input data. This results in some neurons knowing parts of the data better than others. This also means, the bottleneck changes all the time as the bottleneck is created by some of the active neurons.

Denoising autoencoders have a different input than their target output. They are usually used to denoise the input signal. The input data consists of a noised version of the original input data, for example a blurry picture. A denoising autoencoder compares the output to the original, denoised input and determines what has to be improved based on that.

2.3.3 Variational Autoencoders

Another variation on the autoencoder is a VAE. This is used in this project. The basics of a VAE and how it differs from a normal autoencoder is described below.

Autoencoders cannot be used to generate new data, as the latent space is heavily dependent on the distribution of the data and the encoder network. So, one could not just pick a sample from the latent space and decode it to generate new data. A VAE is more generative than a normal

autoencoder because its training is regularized to avoid overfitting and to make the latent space more generative. Like a normal autoencoder, a VAE consists of an encoder, a latent space and a decoder. Likewise, the reconstruction error is supposed to be minimised. However, there a major difference causing the VAE to be more generative. Where a normal autoencoder encoder maps the input to a fixed representation, a vector, a VAE maps the input data into a distribution. The bottleneck is replaced by two vectors: one is the mean vector, the other is the standard deviation vector., as is shown in Figure 3. The latent space of a VAE is sampled from the distribution. Standard Gaussian distribution is often used to compare the output distribution to. For this, Kullack-Leibler (KL) divergence is used, which is a method to measure the difference between two distributions.

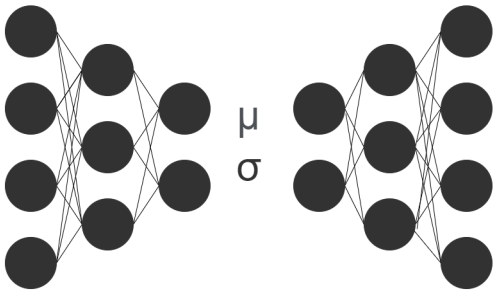

Figure 3: The architecture of a basic VAE

3 State Of The Art Review

This section discusses existing work and compares them to each other. Since the existing work on representation learning with VAEs is limited for AAR, the presented works are divided into sections. First, machine learning for AAR is explored, then feature extraction and deep representation learning methods are discussed for IMU data and lastly autoencoders and VAEs are discussed for HAR and AAR.

3.1 Animal Activity Recognition

In literature, multiple existing ML techniques have been elaborated for AAR. In order to classify animal activities, several types of sensors, classifiers and evaluation methods have been tested and reviewed. The animals in question tend to be domestic animals, such as horses. goats and cows. The activities to be classified vary from eating habits [10], to walking or running. Sensors have been placed on the animals themselves or in their environment, e.g. cameras.

Bocaj et al. [10] use accelerometer data from animals as input data for seven slightly altered Convolutional Neural Networks (CNN). The alterations vary in the type of filters and the number of layers of the CNN. The CNNs outperformed other classifiers, like Naive Bayes [11]. The CNN that used late sensor fusion performs most accurate, 96%.

Kamminga et al. [6] investigate different types of feature extraction techniques for AAR. This is done for the same dataset as this research. Here, the engineered representations are extracted, in time and frequency domain. Additionally, Principal Component Analysis (PCA), a deep net of sparse autoencoders and a Convolutional Deep Belief Network (CDBN) are used to learn the representations of features. From this, the deep net of sparse autoencoders performed best.

In [12], Casella et al. use smartwatches, placed on a saddle of horses and on a rider's wrist, as accelerometer data for AAR. The collected data is classified using four ML algorithms, namely Neural Networks, Decision Trees, KNeighbors and Support Vector Machines. These algorithms were tested and compared to investigate the effects of the algorithms on AAR. All four ML algorithms performed the same for tested sampling frequencies. Support Vector Machines showed to classify accurately, 91%, for the saddle data. The dataset from the rider's wrist was too small to give good results.

In [13], accelerometer data from dogs is collected and classified in an Internet of Things app. The data is from domestic dogs and collected through an accelerometer on a collar. The data is classified using a JRip algorithm, which creates classifying rules that are applicable to the whole dataset [14]. The classifier performed well as it managed to classify with an accuracy of 93% [13].

3.2 Feature Extraction And Representation Learning

A big part of AAR is feature extraction. Using feature extraction, the datasets representations are extracted or learned. This often results in a reduced dimensionality of the data which can be more manageable. For this project, statistical feature extraction and representation learning are used. The relevant existing works using these methods for IMU datasets are presented below.

A method for feature extraction is by using statistical measures. In [15], statistical signal measures are used to classify drastically different human activities. These features are not learned, but just

extracted. However, this is not precise enough when separating activities like walking upstairs from walking downstairs. Therefore, power spectral density was used to estimate the power of the signal in the frequency domain. Auto-correlation recognizes low pitch fundamental frequencies to help distinguish activities even more. An accuracy of 96% was reached using this method.

Shoaib et al. [16] also used statistical techniques to reduce the artifacts in signals from IMU body sensors, that are worn on loose clothing. Here, Kalman filters and unscented Kalman filters are used to get statistical features of the data. Next to the statistical techniques, Shoaib et al. also tested a representation learning method for the same objective. A deep neural network was used to learn the representations of the sensor data. The deep network performed much better on recognizing gestures and locomotion activities.

Another example of deep representation learning in human activity recognition is by Odongo et al. [8]. They use a Long Term Short Term (LSTM) neural network to learn the representations of raw IMU data. These representations can then be used to classify activities like walking, sitting and climbing upstairs or downstairs. LSTM remembers values that seem important and compare new values to the memory (long term) to see if they actually add something or whether they should be forgotten (short term) [17]. This method allowed for using fewer spectral features. Applying this representation learning method to the IMU data gave an accuracy of 89% [8].

In [9], a hybrid framework of CNN and Gated Recurrent Unit (GRU) is used to attempt the learning of representations of activities that are very similar when looking at the raw IMU data. The network consisted of a combination of CNN and GRU layers. The framework showed an accuracy of 89%, which was lower than expected. This is due to the fact that the input data contained very similar activities. The framework performed better than other simpler feature extraction methods from [9].

The feature extraction methods above were all for raw IMU data, but not specifically for AAR. Thus, it cannot be concluded that any of these methods work as well for AAR, although they seem to extract features from IMU data very well. Deep representation learning networks perform better than simpler feature extraction methods, like the statistical signal measurements. The networks that are used for representation learning perform best when adapted to their input data. Even though these methods are not for AAR, they are for HAR, which is somewhat similar as described previously. They therefore might suggest that they are good methods for the dataset of this research as well.

3.2.1 Autoencoders

Common neural networks used for machine learning and unsupervised representation learning are autoencoders and variational autoencoders. These can also be used for AAR and human activity recognition. In literature, autoencoders and VAEs are not extensively explored for AAR, so mainly HAR will be elaborated.

In [7], three methods of unsupervised feature learning are introduced for HAR using sensor data: sparse auto-encoders, denoising auto-encoders and PCA. The performance of these methods was tested on a public HAR data set and compared to existing feature extraction methods. The sparse auto-encoder consists of a neural network with only one hidden layer, with a sparsity constraint. The denoising auto-encoder works like a normal auto-encoder, but additionally it tries to construct a denoised input from the normal input, which is likely corrupted by noise. The PCA reduces

the dimensionality of the input data, by looking at which values are correlated. The reduced dimensionality should represent the data with minimal information loss. From evaluating the three methods, the sparse auto-encoder performed best.

Although autoencoders seem to perform well, they are prone to overfitting in the latent space. They are trained to encode and decode with as little loss as possible, which does not give any specifications or restrictions to the latent space. When an autoencoder is overfitted, some points of the latent space will produce senseless content after it is decoded. To avoid the overfitting, the latent space should be regularized. This is the case in VAEs.

3.2.2 Variational Autoencoders

Anazco et al. have built a VAE to denoise IMU signals to improve HAR in [4]. The VAE denoises accelerometer and gyroscope signals. These denoised signals were then classified. The input data is encoded into a representation vector using three convolutional layers, as well as three LSTM layers. The latent space consists of two dense layers. The decoder has the same layers for decoding. The VAE successfully denoised the raw IMU data with 9dB. This improved the HAR by 6%.

The Motion2Vector [5] is a HAR VAE model, that learns a representation of a timed period of unlabeled human activity data. This VAE creates an encoded vector of raw accelerometer and gyroscope data. A LSTM layer is used in this VAE to learn the representations. The model was tested on existing human activity sensor data, e.g. The Heterogeneity Human Activity Recognition dataset, and on in-lab created human activity data that fits the model precisely. The model outperformed hand-crafted features. The Motion2Vector performed better when limited labeled data was available.

The last approach at feature extraction using autoencoders for AAR is in [6], where the same horse dataset is used as the one that will be used for this project, as well as a goat dataset. Here, a PCA, Convolutional Deep Belief Network (CDBN) and a deep net of sparse autoencoders are used for the representation learning of the raw IMU data. For the feature extraction, time and frequency domain methods were also tested. The CBDN is a generative model with multiple hidden layers, that is trained using greedy layer-wise training. The sparse autoencoders performed best for unlabeled data that are smaller sized. Since the horses wore only one sensor, opposed to the goat data, who wore multiple sensors, the sparse auto-encoder gave the best results for the horse dataset. The CBDN performed better than the sparse autoencoder for a dataset that is smaller than the horse dataset.

3.3 Discussion And Conclusion

Table 3.3 shows an overview of all methods that were mentioned in the state of the art. The VAE by Anazco et al. performed well for denoising. The VAE helped generalizing the original input data. The sparse autoencoder by Kamminga et al. showed good results, but only for a small dataset. The combination of the two suggests that a generative model that can handle more data is interesting to investigate. VAEs and Generative Adversarial Network (GAN) are such generative model and might be able to handle bigger datasets. From the Motion2Vector, it can be concluded that using a VAE for the learning of the representations actually outperforms feature extraction using statistical features. Especially since the data input is very similar in the Motion2Vector, it seems promising that the addition of a VAE can improve the current framework.

There are many ways autoencoders and VAEs can be applied in activity recognition. Depending on what parameters are important for the outcome, different models might perform better. Autoencoders, like the ones by Li et al. seem promising, but autoencoders are very prone to overfitting. VAEs might thus be a better option. Multiple VAEs have been tested for HAR and give good performance, indicating good results when applying it to the horse dataset. For the horse dataset, sparse autoencoders seem to work well since the dataset is relatively small, although a generative model that can handle bigger datasets could potentially be even better. From the last three papers, it can be concluded that a generative model is potentially a very promising addition to the current framework. For the particular data and goals of this project, a VAE model would be a very fitting generative model.

Paper	***Feature Extraction Technique***	***Description***	***Performance***
Dehganzi and Sahu [15]	Statistical signal measures	Extracting statistical values from the data, like mean, RMS and spectral peak	96% Accuracy
Shoaib [16]	(Unscented) Kalman filters	Optimally estimating statistical values, since they cannot be measured directly. Unscented produces sampling points around the current state	80% F-score
Odongo et al. [8]	Deep LSTM network	Remembers important values and compares new value to see if they are more relevant	89% Accuracy
Siraj et al. [9]	Hybrid network of CNN and GRU	Learns spatial features and remembers the relevant ones	89% Accuracy
Li et al. [7]	Sparse autoencoder	Autoencoders with a sparsity restriction	92% Accuracy
Li et al. [7]	Denoising autoencoder	Autoencoders with denoising input reconstruction	90% Accuracy
Anazco et al. [4]	Denoising VAE with LSTM layers	VAE with denoising input reconstruction	96% Accuracy
Bai et al. [5]	Motion2Vector (VAE)	VAE with LSTM layers	91% Accuracy
Kamminga et al. [6]	SAE (VAE)	Autoencoder with sparsity constraint	90% F-score

Table 1: Overview of feature extraction methods and their results. **Light bue**: Feaure extraction methods, **Medium blue**: General representation learning methods, **Darker blue**: Representation learning with autoencoders and VAEs

4 Approach

In this section, the approach for the project is explained. A pipeline is created in which three different ways of feature extraction are tested; statistical feature extraction, representation learning with a VAE and no feature extraction. The general approach for the pipeline is elaborated and the more specific design decisions and challenges are explained. The specific implementation is unfolded in the methodology section.

4.1 The Pipeline

In order to evaluate whether the addition of a VAE is beneficial to the AAR process, a pipeline is created. Figure 4 showcases this pipeline in its entirety. The steps are preprocessing, feature extraction, classification and evaluation respectively.

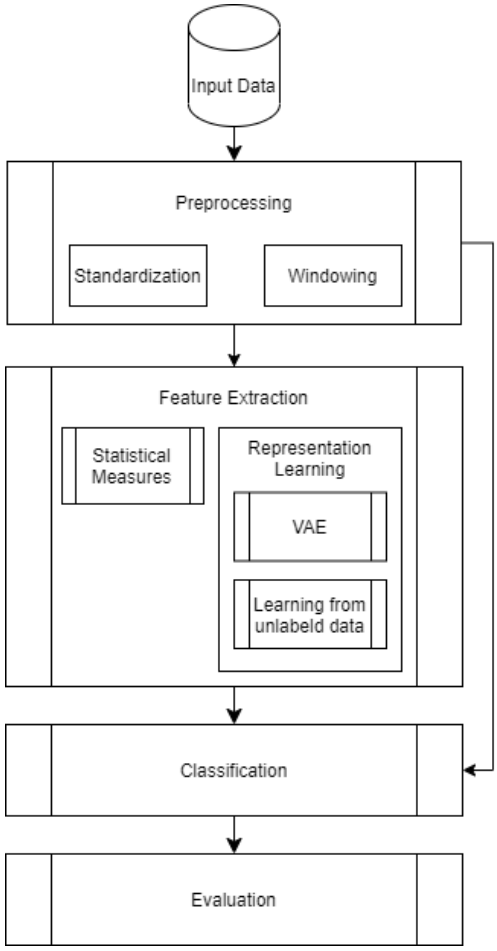

Figure 4: The pipeline with three feature extraction methods

The preprocessing is necessary to get the raw data to fit the network it is going into. For this, the data is standardized to be within a range of 0 to 1, to fit the networks. Then the data is segmented into windows with 50% overlap. These standardized windows will then be used to go to the next step; the feature extraction.

This pipeline consists of three feature extraction methods. The first method uses statistical feature extraction. The second uses representation learning as feature extraction method. This is where the VAE is implemented. The last step uses no feature extraction and will use raw data values. In Figure 4 the three feature extraction methods are displayed next to each other. This also means that the pipeline will be executed three times, each time using one of the feature extraction methods. They are not executed simultaneously. The features that are extracted, will be fed through the classifier.

The classification process consists of training a classifier and then testing it with a test set. The train and test set are created from the extracted features. Using the training set, a vanilla classifier is trained. Then, the test set is used to evaluate how well the classifier managed to label each activity. For this part, labeled data is necessary. Hence, the whole pipeline is executed using labeled data each time.

The classifier is trained the same way each time. The evaluation process also remains the same. The only difference in the classification process is thus the input data. It either consists of statistical features, latent representations or raw data. This makes it possible to compare these methods fairly.

4.2 Statistical Feature Extraction

The first statistical feature extraction method uses statistical measures. For this, statistical values are calculated from the raw data. These statistical values are features of the data that represent the data. In most cases it also reduces the dimensionality of the data. The statistical feature extraction consists of two steps. The first step is t calculate the statistical measures, the second is to implement these into the pipeline.

The statistical feature extraction method takes statistical features of the sensor data. Since this step is done after the preprocessing. The features are then fed into the classifier. The measures extracted from the data are the mean, variance, standard deviation, median, 25^{th} and 75^{th} percentile, maximum, minimum, kurtosis and skewness. These measures are chosen since they are comparable to the measures by Kamminga et al. [6]. That research uses the same dataset as this project. For fair evaluating purposes similar features should be extracted. An overview of the statistical measures that are extracted can be found in Table 2. These features are then fed to the classifier, which is the next step of the pipeline.

Statistical measure	*Description*
Mean	Average value of window
Standard deviation	Amount of deviation from values in window
Median	Middle value of window
25^{th} percentile	The value below which 25 % of the observations are found
75^{th} percentile	The value below which 75 % of the observations are found
Maximum	The highest value of the window
Minimum	The lowest value of the window
Kurtosis	The degree to which values peak together
Skewness	The asymmetry of the distribution of the window

Table 2: Overview of statistical measures extracted as features

4.3 Representation Learning With A Variational Autoencoder

The second feature extraction method uses representation learning. Here, instead of calculating features from the data, a network learns the representations of the data. As was shown in the state of the art, a VAE promises to be well suited for this. Consequently, a VAE will be used for this step. Putting a VAE into the pipeline consists out of three steps. The first step is to design the VAE. The second step is to train said VAE. And the last step is to implement the trained VAE into the pipeline.

4.3.1 Designing The Variational Autoencoder

A VAE always looks the same in the sense that there is an encoder, a decoder and a bottleneck; the latent representations. However, several decisions have to be made to create a suitable VAE. For this pipeline, a VAE is built that takes an input size that is the same size as the windows of the preprocessed data. The bottleneck is created with a size of three for plotting and showcasing purposes. The number of layers in the encoder and decoder is based on the input size and how fast it should reduce in size to get to the latent representations. In this case the input is halved each time until it reaches the size of the latent representation.

4.3.2 Training The Variational Autoencoder

Before the VAE can be used in the whole pipeline, it has to be trained. This training process is shown in Figure 5. The data that is used for this step is altered, which is explained in the methodology. First, all data is preprocessed to create the correct input shape for the VAE. This preprocessing is identical to the preprocessing step from the pipeline. The preprocessed data goes through the encoder. This is a network that compresses the input data into a mean and covariance. From this a vector is created which represents the input data. This is called the latent representation. This latent representation can go through the decoder. The decoder tries to reconstruct the input data, given just the latent representation. Because the output of a VAE should be similar to the input data, the decoder network in similar to the encoder network, except its reversed.

The training process is to train the encoder and decoder network. The training process is steered by the loss function. This looks at how similar the output data is to the input data and adjusts the network accordingly. Although the decoder will not be used once the trained model is added

to the pipeline, it is still mandatory to train it. This is because the decoder outputs the generated data. The loss function requires that data to compare it to the input data to see how well the VAE managed to generate the data. Based on this it will steer the representation learning process.

A mathematical approach to the training process starts with the preprocessed input data as x. When training, the encoder compresses x to $e(x)$. This has mean μ and covariance σ, with a normal distribution $N(\mu_x, \sigma_x)$. Both μ and σ are used to create a sample vector z. From this, z is created to go through the decoder. Considering the output should closely resemble the input data, $x \approx d(z)$. A loss function would then be defined as $loss = |d(z) - x|$. This gives the absolute difference between the input and output of the VAE.

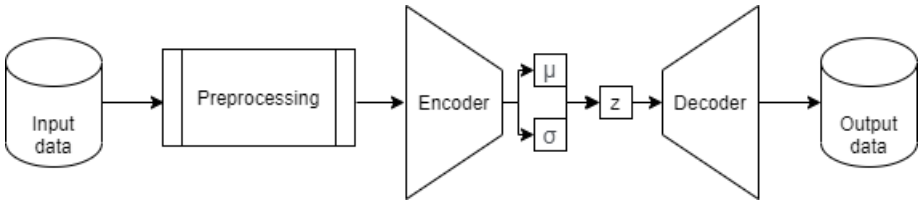

Figure 5: The training process of the VAE with mean μ, covariance σ and latent sample z

4.3.3 Implementing The Variational Autoencoder Into The Pipeline

When implementing the trained VAE in the pipeline, some elements need to be considered. When training the classifier with the features extracted in the VAE, only the latent representations are needed. This means only the encoder is necessary. This is showcased in Figure 6. The data that goes through the trained VAE model is preprocessed the same way as each other part of the VAE. The data is windowed and standardized. Each window goes through the encoder of the VAE and the latent representations are recorded. Since the VAE reduces the dimensionality of the data, the input shape of the classifier changes and thus the classifier needs to be adapted to that.

Considering the classification process needs to be evaluated, this process is executed with labeled data. The train and test set are created and preprocessed. Each window then goes through the encoder and the new windows consist of the latent representations of the original window. The train set is used to train the classifier and the test set is used to evaluate the process. The labels are only used to evaluate the classification. For the feature extraction the labels are there, but they are not used, since it is an unsupervised representation learning method.

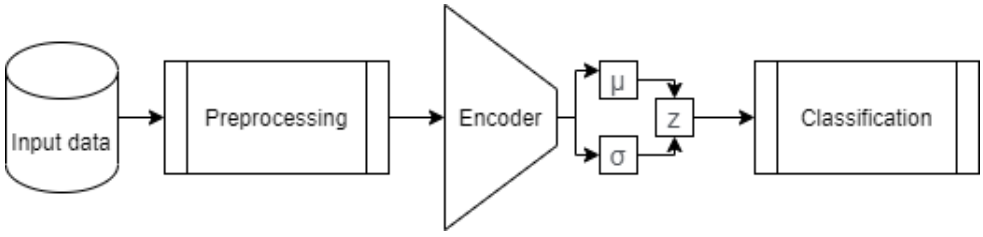

Figure 6: The VAE implemented in the pipeline

4.4 No Feature Extraction

The last method of feature extraction uses no feature extraction. The data is preprocessed the same as each other step in the pipeline and from this data no features are extracted. The windows are fed to the classifier just like the other feature extraction methods. The labeled data is used because the classifier needs the labels to evaluate the classification.

4.5 Evaluation Process

The pipeline can be evaluated with each type of feature extraction method. For this, the three feature extraction methods are executed. The classifier is then also trained three times. Each time with the extracted features. For this, a test set of labeled data is used each time. Figure 4 showed the entire process. It is separated into three parts, as described above. This way, the results can be compared to address whether the addition of the VAE is beneficial to the classification of animal activities.

For each feature extraction method, the classifiers output is compared to the actual labels of the data to see how well it performed. This is why labeled data is used when testing the pipeline. The performance of the classifier for each feature extraction method is presented in the form of a confusion matrix, as well as the calculated statistical measures from the performance. This makes the evaluation process applicable and reliable.

The performance of the classifier will be evaluated by looking at the accuracy, F-score and Matthews Correlation Coefficient (MCC) score. The accuracy describes how close the labels from the classifier are to the actual labels. This looks at the true positive and true negatives. The accuracy will be shown since this is most intuitive and easy to understand. However, the accuracy might not reflect all information and thus the F-score will also be looked at. The F-score also describes how close the classifiers output is to the actual labels, but this prioritizes the false positives and the false negatives. The F-score is also more fitted for an imbalanced dataset [18]. The downside of accuracy and F-score is that they look at either the true and false positives or the true and false negatives. A less biased score is therefore a combination of the two, the MCC score [19]. The MCC score takes all categories of the prediction into consideration and might therefore be considered as a less biased observation. The formula for calculating these scores can be found below:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$F - score = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

By comparing the results of the classifier in the pipeline with statistical feature extraction, the VAE and no feature extraction, a conclusion can be made about the addition of the VAE to the pipeline.

Another conclusion can be made on the addition of feature extraction in general. It can also be concluded whether the VAE is a more valuable feature extraction method than using statistical measures in AAR.

5 Methodology

In this section the specifications of the projects design are explained. First, the dataset is introduced as it is provided by Kamminga et al. [20], as well as the adaptations made for this project. Next, each step of the pipeline is demonstrated. Lastly, the exact tools that are used in the project are mentioned.

5.1 Dataset

The data used in this Graduation Project has been gathered by Kamminga et al. [20]. This dataset consists of data gathered from 18 different horses and ponies across a period of seven days, during which the horses participated in both riding and free roaming activities throughout their pasture. The animals wore an inertial measurement unit (IMU), containing an accelerometer, gyroscope and magnetometer, in a collar. These IMUs made use of a 100 Hz sampling rate, recording a total of 1.2 million data samples, each describing a 2 second interval, by the end of the week.

As the collar containing the IMU can still slightly move and rotate around the animal's neck, the dataset also includes l2-norm values for each of the sensors. These values can be used to compensate for any recorded movement of the collar that does not correspond to movement related to the horse's activity.

The data as it was published by Kamminga et al. [20] consists of labeled and unlabeled data. The used labeled data as a whole is not equally labeled; only data from 11 subjects were labeled, of which five subjects were labeled more extensively, which can be seen in Figure 7. The dataset contains 18 activities, which can be found in Table 3. Again, some activities are more extensively labeled than others, which can be seen in Figure 8. The dataset is thus rather imbalanced, which should be kept in mind for training the VAE and the classifier.

For this project, the more extensively labeled subjects and activities are used in attempt to tackle the imbalance of the dataset. For the subjects, the five more extensively labeled subjects are Galoway, Patron, Happy, Driekus and Zafir. For the activities, trotting-rider and trotting-natural are combined, as well as for the running-rider and running-natural activities, since these are similar activities of which not both activities contain enough samples to be used in this project. Thus, only data from the horses Galoway, Patron, Happy, Driekus and Zafir will be used for the activities walking-rider, trotting (rider and natural), grazing, standing, running (rider and natural) and walking-natural.

The remaining dataset contains 9403903 labeled (unwindowed) samples.

Activities	***Description***
Standing	Horse standing on 4 legs, no movement of head, standing still
Walking natural	No rider on horse, the horse puts each hoof down one at a time, creating a four-beat rhythm
Walking rider	Rider on horse, the horse puts each hoof down one at a time, creating a four-beat rhythm
Trotting natural	No rider on horse, two-beat gait, one front hoof and its opposite hind hoof come down at the same time, making a two-beat rhythm, different speeds possible but always 2 beat gait
Trotting rider	Rider on horse, two-beat gait, one front hoof and its opposite hind hoof come down at the same time, making a two-beat rhythm, different speeds possible but always 2 beat gait
Galloping natural	No rider on horse, one hind leg strikes the ground first, and then the other hind leg and one foreleg comedown together, the other foreleg strikes the ground. This movement creates a three-beat rhythm
Galloping rider	Rider on horse, can be right or left leaning, one hind leg strikes the ground first, and then the other hindleg and one foreleg come down together, the other foreleg strikes the ground. This movement creates a three-beat rhythm
Jumping	All legs off the ground, going over an obstacle
Grazing	Head down in the grass, eating and slowly moving to get to new grass spots
Eating	Head is up, chewing and eating food, usually eating hay or long grass
Head shake	Shaking head alone, no body shake, either head up or down
Shaking	Shaking the whole body, including head
Scratch biting	Horse uses its head/mouth to scratch mostly front legs
Rubbing	Scratching body against an object, rubbing its body to scratch itself
Fighting	Horses try to bite and kick each other
Rolling	Horse laying down on ground, rolling on its back, from one side to another, not always full roll
Scared	Quick sudden movement, horse is startled

Table 3: Overview of horse activities [21]

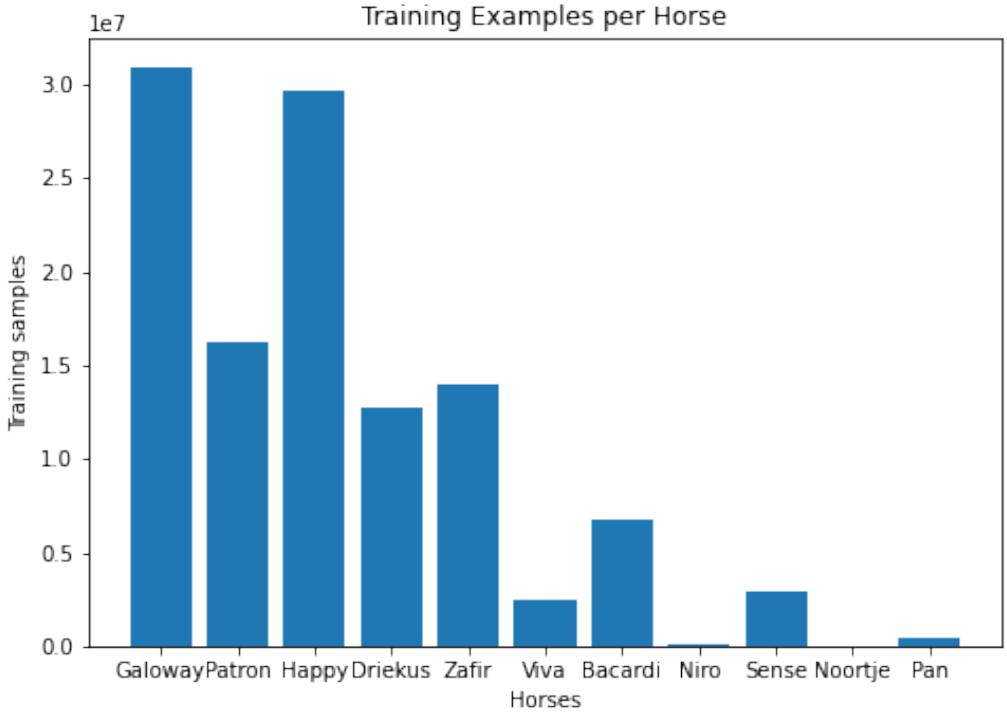

Figure 7: The distribution of labeled samples over the different horses.

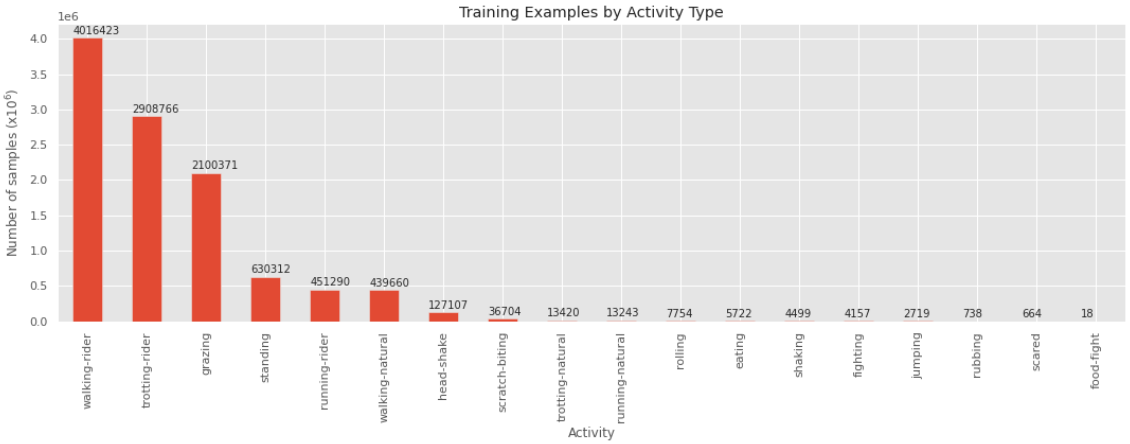

Figure 8: The distribution of labeled activities. [20]

The data is contained in CSV files, describing the x, y and z values of the accelerometer, gyroscope and magnetometer. Next to that, the subject, segment, label and date and time are denoted.

In Figure 9, 10 and 11 one data sample can be found for the activities eating, running-natural and shaking.

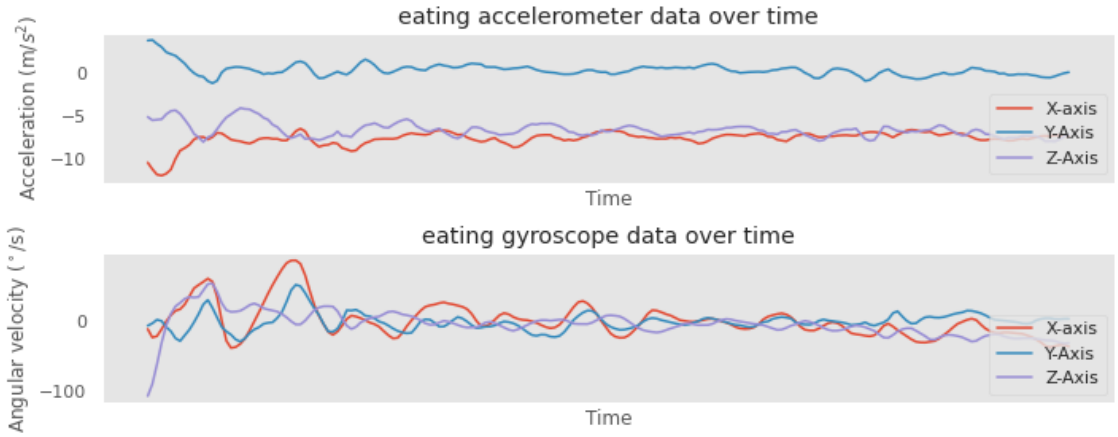

Figure 9: An accelerometer and gyroscope measurement of a horse eating.

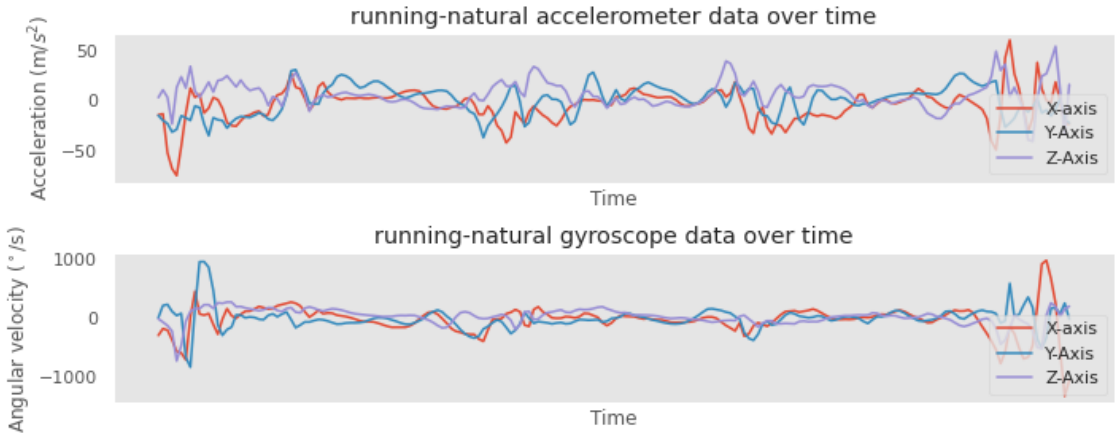

Figure 10: An accelerometer and gyroscope measurement of a horse running naturally (without a rider).

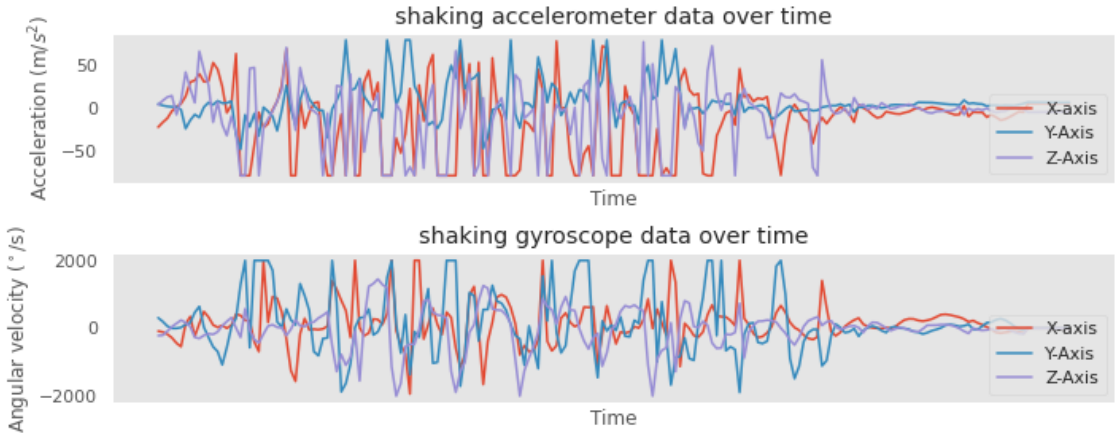

Figure 11: An accelerometer and gyroscope measurement of a horse shaking.

5.2 Training The Variational Autoencoder

Before the pipeline is built for each feature extraction method, the VAE is trained. This has to be done preliminary to the pipeline, seeing that only a trained VAE model can extract the features.

Preprocessing data for training of the VAE The VAE is trained on 5 different combinations of the data; the labeled data of 5 subjects, the unlabeled data of 5 subjects, the labeled and

unlabeled data of 5 subjects, all unlabeled data and all unlabeled data with the labeled data. For each combination, the CSV files of the subjects are combined into one pandas dataframe and whilst doing so almost all columns are removed from the raw unlabeled data, so only the A3D magnitude vector of the accelerometer is left. The other columns are dropped to match the input data of the classifier. The remaining dataframe is windowed, to windows with a size of (200,1) with 50% overlap. Then, the windowed dataframes are converted to tensors and the values are standardized to a scale of `(0,1)`, by adding the lowest relevant value of the column to all values to make each value in the column a positive number.

Building the VAE The VAE consists of an encoder and a decoder. The input size of the VAE is 200, since this is the size of the preprocessed windows. The encoder consists of six layers that each compresses to halve the previous size. The sixth layer converts the data into the latent representations μ and σ. These are reparametrized into vector z. z has a latent size of three and is used as an input of the decoder. The decoder is reverse symmetrical to the encoder.

Training the VAE The training of the VAE uses the preprocessed unlabeled data. A batch size of 512 is used. The loss function is calculated for each value and the outcome steers the training of the VAE. The loss function uses `binary_cross_entropy()` from PyTorch. Lastly, the `Adam()` optimizer function using gradient descent is applied to minimize the reconstruction error.

5.3 Feature Extraction

Three feature extraction methods are tested. These feature extraction methods are part of the pipeline, but each time only one of the methods is implemented. They are all implemented the same way, except that the manner of getting the representations changes. These methods are described here and they are referred to in the classification pipeline explanation.

Statistical feature extraction The statistical feature extraction is performed per window that is created from the data. The statistical values that are calculated are the mean, standard deviation, median, 25^{th} and 75^{th} percentile, maximum, minimum, kurtosis and skewness. These are calculated along A3D the column of the dataframe. For this, Numpy functions are used since the windows are also created as Numpy arrays. Scipy functions are used for the kurtosis and skewness.

Representation learning The representation learning is done using the trained VAE model. The features are extracted from the latent space and thus only the encoder is required. To get the latent representation, the windows that consist of Numpy arrays, first have to be converted PyTorch tensors to fit the trained model. After this, each window goes through the encoder and the three latent representations are recorded. From these latent representations, the new window is created. This window is then converted back to a Numpy array to fit the classifier again.

No feature extraction For this feature extraction method, no features need to be extracted. The data is already in the correct shape for the classifier, so no additional steps are taken for this method.

5.4 Horse Activity Recognition Pipeline

The pipeline is split up into four classes: preprocessing of the data, the feature extraction, the database interactions, and the main class, which also contains the classifier.

Dataset Usage For this project, both labeled and unlabeled data are used. As was explained in the approach, the project consists of three phases. Each of these phases requires different data, for different reasons. The training of the VAE requires unlabeled data and the rest of the project uses labeled data. For the training process of the classifier with VAE the labels are not used when the data goes through the VAE, but they are used when training the classifier itself. Training the classifier for the classifying process without VAE, labeled data is required. For the evaluation process, the test set consists of labeled data. The data goes through the entire pipeline without labels, but the labels are needed to compare the classifier results with the real labels.

Preprocessing data for classifier A sub-selection of horses is made as for some horses there was comparatively little data available. The selected horses are Galoway, Patron, Happy, Zafir, and Driekus. The corresponding data sets are combined into a single dataframe and rows where values are missing are removed.

Sensor selection The three columns describing the various magnetometer axes are dropped, as this data was found to be too prone to alterations as a result external disturbances, such as magnetic fields, and thus unreliable as a whole. Additionally, the l2-norms of the various sensors are also not used during the study, and as such are removed from the feature set. Only the A3D value of the accelerometer data is used as sensor data and all other IMU sensor values are dropped. The gyroscope is dropped since the values are unreliable considering the sensor was worn on a loose sensor necklace around the subjects' necks. Only A3D is used, since it concisely describes the x, y and z axis of the accelerometer. This results in the following feature set, seen below in Table **??**.

Feature	***Description***
A3D	3D magnitude vector of the accelerometers x, y and z values
label	Label that belongs to each row's data
segment	Each activity has been segmented with a maximum length of 10s. Data within one segment is continuous. Segments have been numbered incrementally.
subject	Subject identifier

Table 4: Overview of the feature set. Adapted from [20, p.4]

Data filtering The accelerometer measurements are inherently noisy. Thus, it is important to filter out high-frequency noise from the measurements. This is done with a low-pass Butterworth filter with a cut-off frequency of 30 Hertz. Arablouei et al. [22] show that the most power in the signal's ranges from 0 to 25 Hz, making 30Hz a reasonable cut-off frequency for high-frequency noise.

Splitting the data The dataset is split into a training and testing subset using the Leave One Out Cross Validation method. This method was also used by Kamminga et al. [6] for the same

dataset, to address heterogeneity. During the testing phase, one of the five horses is selected as the testing subject and the other four horses are included in the training subset. For example, if Galoway is used as a test subset, then Patron, Happy, Zafir, and Driekus are included in the training subset. These steps are performed for all five horses, therefore testing is done five times.

Feature scaling Since some of the values are of a much larger size than others, it is important to scale them so that they are easily comparable. To do so, the accelerometer samples are divided by the highest value within the corresponding axis to obtain normalized values between 0 and 1.

Windowing Windowing is done with a sliding window of two seconds with 50% overlap, so a step distance of one second. This matches sampling frequency of the IMU sensor, which is 100 Hz. These windows can now be used as data instances for feature extraction.

Feature extraction The feature extraction is done for each window, hence it is done after the windowing. When running the pipeline, each time only one of three feature extraction methods are used. These methods are described above. They all take the windows and extract the representations per window. Then, the same number of windows are returned, but now the feature extraction method has been applied to it. The shape of the returned windows is `(200,6)`, `(200,3)` and `(200,1)` for the statistical feature extraction, representation learning and no feature extraction respectively. The input size of the classifier is adapted to this accordingly.

Shuffling and reshaping Once the features are extracted, all new windows are shuffled with the use of the `shuffle()` method from scikit-learn. After shuffling, the training data is reshaped into a one-dimensional array to fit into the classifier.

Encoding labels In order to make the dataset more suited for most machine learning algorithms, the categorical labels should be converted into numerical ones. However, as there is no ordinal relationship between the original categorical labels, one-hot encoding should be applied to any numerical label representation to avoid the algorithm potentially trying to make use of any non-existent ordinal relationship.

To do so, the various activity labels are first converted into numerical labels using the `LabelEncoder()` function provided within scikit-learn's preprocessing library. Following this, one-hot encoding is applied to the integer representation of the labels. The resulting encoded labels are added as an extra column to the dataframe.

Classifier The classifier used is a sequential classifier from keras, which represents a neural network with multiple layers. The first layer is a Reshape layer, where the training data is reshaped from 1200 rows to 200 rows per batch, so it fits the input of the classifier. Next, three Dense layers are added, representing three hidden layers in the neural network, each with 100 fully connected nodes. The activation function used in each of these layers is a rectifier, which is the ReLu activation function. After the three hidden layers, a Flatten layer is added to flatten the data. The last layer is the output layer, which is a Dense layer with the same number of nodes as the number of activities and a softmax activation function.

During training, a batch size of 600 is used each iteration and has a total of 50 epochs. After training, an evaluation is performed and a confusion matrix is constructed from this data.

5.4.1 Testing The Classifier

The testing phase goes in three ways: Once for each feature extraction method. As mentioned above, both training and testing phases are performed five times, once per tested horse. For the testing phase the steps are performed in the same order as described above: splitting the data, feature scaling, windowing, getting the representations, shuffling, reshaping, encoding, training the classifier and lastly, testing.

Firstly, the sensor values in the test data are standardized, after which windowing, feature extraction and reshaping is performed. The only difference in the preprocessing of the test data and the training data, is that the test data does not get shuffled.

Lastly, the classifier model is tested with the keras `predict()` method. Per horse, the experiment performance is saved in the database. Next to that, the performance of each activity per horse is also saved in the database.

5.5 Tools

In this project multiple tools are used, which will be described below.

5.5.1 ITC Geospatial Computing Portal

The ITC Geospatial Computing Portal from the University of Twente is used to run the code on [23]. This portal allows, among other programming languages, Python 3 code through the JupyterLab environment [24]. Apart from running the code, the CSV files containing the raw IMU data can also be stored on this server.

5.5.2 Programming Language And Packages

For this project Python 3 is used with a couple of packages.

Pandas The Pandas package supports data analysis and data manipulation, allowing the user to retrieve and shape the data [25].

Numpy The Numpy package allows mathematical calculations [26]. In this case, these calculations are used for conversions between number types (integer, float, etc.), retrieving maximum values and shaping arrays.

PyTorch The PyTorch package allows for tensor computations and deep neural network models built on a tape-based autograd system. [27]

Scikit-learn The scikit-learn (or sk-learn) library is made for predictive data analysis [28]. In this project, it is used to define the metrics and for preprocessing.

Kras The keras package focuses on Deep Learning, including methods for implementing a Deep Learning algorithm [29].

Pewee The peewee package is used to implement a database to store and retrieve the results of experiments [30].

Seaborn Seaborn is used for data visualization [31]. In this case, seaborn is used to plot a confusion matrix.

Other packages Other packages used are scipy, glob, matplotlib, uuid, IPython, datetime, re and ast.

5.6 Database

Using the Python package Peewee, a database is created to easily store experimental data. This database is connected to a virtual server. There are two tables, one for the experiment results per horse and one for the experiment results for a specific activity per horse.

The experiment table consists of several columns, containing a summary of the results from all activities:

- Key: a unique key is generated per experiment with the uuid Python package.
- Username: The name of the person who performed this experiment.
- Horse: The name of the horse which is in the test dataset.
- Date: The date on which the experiment was conducted.
- Accuracy: The overall accuracy of this experiment.
- Balanced accuracy: The overall accuracy of this experiment, balanced on the number of samples per class.
- F-score: The overall weighted F-score of this experiment.
- MCC: The overall MCC of this experiment.
- Recall: The recall of this experiment.
- Confusion matrix: The confusion matrix of this experiment.
- Parameters: The parameters used for this experiment, including Time Periods, Step Distance, Epochs and Batch size.
- Description: A description of the specific settings used for this experiment. This can, for example, entail of the structure of the classifier. The description is added in text-format.

The activity table also has several columns, focusing on each activity within an experiment:

- Key: A unique key, corresponding to the experiment.
- Horse: The name of the horse which is in the test dataset for this activity.

- Activity: The name of this activity.
- Accuracy_activity: The accuracy of this activity.
- Recall_activity: The recall of this activity.
- Specificity: The specificity of this activity.
- Precision: The precision of this activity.
- TP: The number of times the system predicted Positive and the actual value was also Positive (True Positive).
- TN: The number of times the system predicted Negative and the actual value was also Negative (True Negative).
- FP: The number of times the system predicted Positive and the actual value was Negative (False Positive).
- FN: The number of times the system predicted Negative and the actual value was Positive (False Negative).

6 Results And Evaluation

The results are split up into three sections. Firstly, the results of the classifier using statistical feature extraction are shown. Then the representation learning results are displayed. Lastly, the results of the classifier without any feature extraction are presented.

6.1 Overview Of The Results

This section shows the results of the classifier that is trained with statistical feature extraction, the classifier trained with latent representations of a VAE and the classifier trained without any feature extraction. The main results are combined into a table and they are shortly elaborated. These results are elaborated on in the next sections.

Each time the classifier has been trained, it has been trained with the labeled data of five subjects, as explained in the methodology. The results shown are the combined results after training and testing the classifier five times alternating each subject as a test set. The classifier has been trained on 25 epochs, with a validation split of 0.2. Because of this validation split it did not always reach the 25 epochs.

The classifier has been trained using the three different feature extraction methods. Each of these methods have been performed using the labeled data of five subjects. The results from Table 5 show the accuracy, the MCC score and the F-score. From this table it can be deduced that the classifier trained on the statistical features performed best in all measures. The accuracy showed the highest score followed by the MCC and then the F-score. It can also be presupposed that the classifier that is trained with the latent representations performed the lowest out of all three feature extraction methods. This means it also classified poorer than the classifier that uses no feature extraction.

Feature extraction method	***Accuracy***	***MCC***	***F-score***
Statistical feature extraction	**0.86**	**0.85**	**0.81**
Representation learning on labeled data of 5 subjects	0.68	0.78	0.73
Representation learning on unlabeled data of 5 subjects	0.61	0.56	0.50
Representation learning on labeled data and unlabeled data of 5 subjects	0.71	0.75	0.66
Representation learning on all unlabeled data	0.70	0.78	0.76
Representation learning on all data (labeled and unlabeled)	0.61	0.56	0.50
No feature extraction	0.81	0.80	0.79

Table 5: Results of training the classifier using three different feature extraction methods

Below is a more in-depth description of these results.

6.2 Variational Autoencoder Model

The VAE models had to be built and trained before they were implemented in the pipeline. To properly evaluate the implementation of the models in the classification pipeline, the VAEs should

perform well by itself. For this, several parts of the training process are recorded to find out how well a model is trained.

6.2.1 Training Results Of The Variational Autoencoder Models

The training process of a VAE can be displayed by plotting the loss from the loss function throughout the process. This loss is calculated in the model by comparing the output data to the input data using binary cross entropy. As can be seen in Figure 12, the losses from each trained model are very similar. It makes sense that the loss from the trained models is similar, since the only difference in the training process is the input data. It can also be seen that each time the loss immediately drops to a value around 0.3 and does not improve much afterwards. In fact, the models seem rather unstable. This can be deduced by the peeks that are present in Figure 12 for each model. There are some iterations, where an attempt to better the training process actually made the performance worse. These mistakes are also immediately corrected.

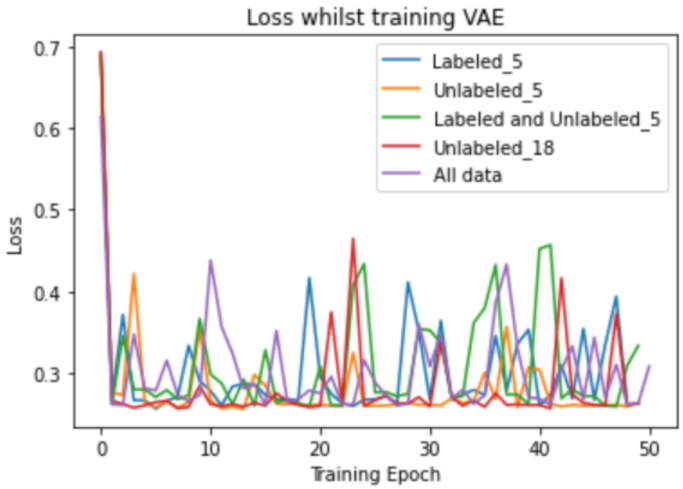

Figure 12: The loss function whilst training the VAE. **Labeled_5**; Trained on labeled data of 5 subjects. **Unlabeled_5**; Trained on unlabeled data of 5 subjects. Labeled and **Unlabeled_5**; Trained on the labeled and unlabeled data of 5 subjects, **Unlabeled_18**: All unlabeled data, **All data**: All unlabeled and labeled data

6.2.2 Investigating The Trained Variational Autoencoder Models

The trained VAE models can also be investigated before they enter the pipeline. This way, it can be fairly evaluated whether the VAE is an adequate addition to the pipeline or not.

To investigate the models, the latent representations are plotted. The latent space is the bottleneck of the VAE model, which is where the input data is compressed the most. These latent representations are later taken as the extracted features for the classifier. Considering the latent representations have a size of three, they can be plotted in a 3D plot, like in Figure 13. Since

the latent representations, represent the activities from the input data, ideally the plotted representations should be separated into groups. One group for each presented activity. To investigate this, labeled data was fed through each trained model and the latent representations were recorded. Using the representations and the labels, the latent representations are plotted and color coded. The results of this are shown in Figure 13. Figure 14 shows the same plots, but zoomed in until the other groups were more most visible.

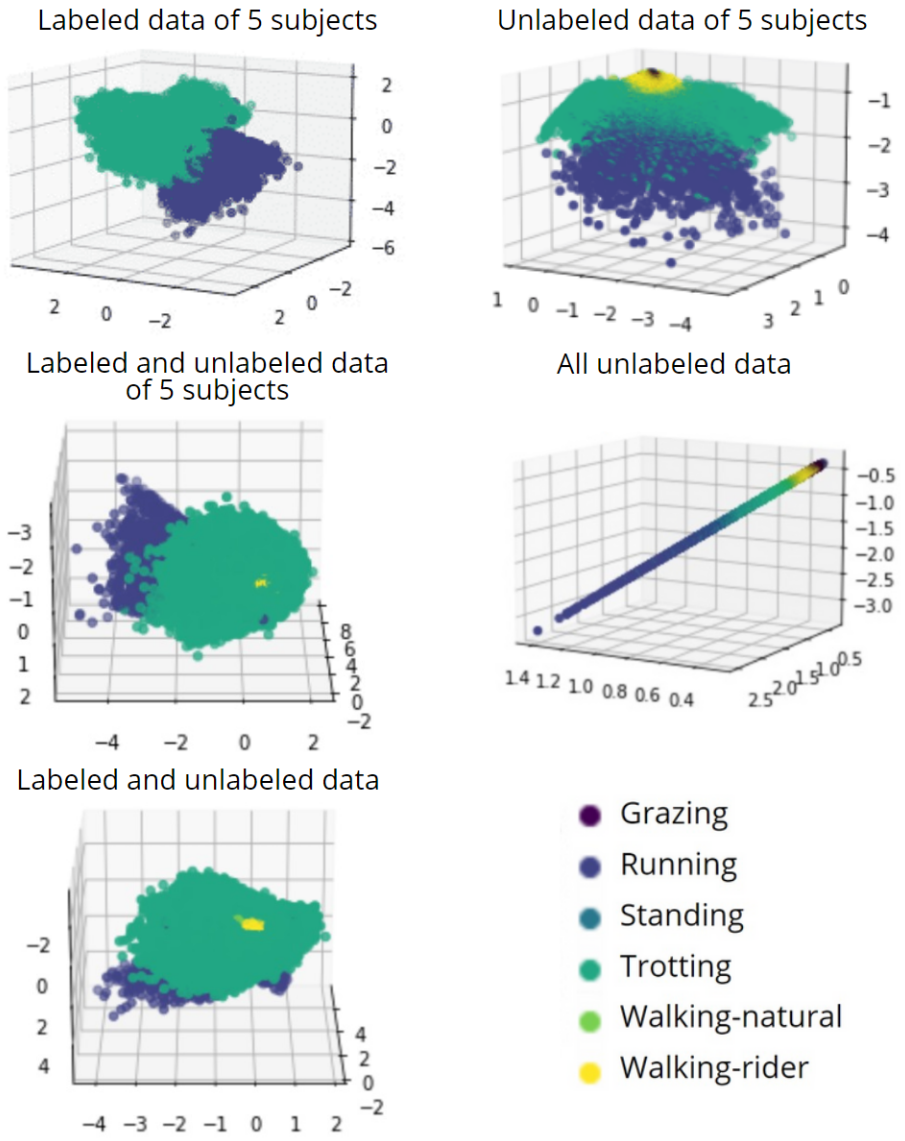

Figure 13: The 3D plot of the latent representations of trained VAE models with varying input data

Labeled data of 5 subjects From Figure 13 it can be seen that there are two groups visibly separated. These represent *trotting* and *running*. The other activities are not very visible when looking at the plot from this angle, but Figure 14 shows that there are more groups in the middle of the plot. Here, *walking-rider* is shown more clearly. Although not visible, it is possible that the activities *standing* and *grazing* are in the middle of the yellow group. The activities that are faster by nature, and thus have higher acceleration, seem to be more scattered and on the outside. The slower activities are more clustered towards the middle. There is less separation between these slower activities, but considering these activities are similar by nature, that is probably reflected in the input data. Hence, also in the latent representations. *Grazing* and *standing* are very similar, since they both involve standing still. *Trotting* and *running* however, differ in speed and have different accelerations. The VAE is likely able to better separate this.

Unlabeled data of 5 subjects Figure 13 shows that there is definitely a distinction between most activities for the model trained on unlabeled data of 5 subjects. Especially *running*, *trotting* and *walking-rider* are visibly separated. However, the other activities are more uniform. Figure 14 shows a zoomed-in version of the same plot and there it can be seen that the top part is mainly dominated by *grazing*. There are a few dots of *standing* and *walking-natural* as well. Again, the nature of the activities likely plays a big role in the separations from Figure 14. There are several dots of *walking-natural* within the area of *walking-rider*. Both these activities describe a walking horse. It would make sense if the VAE confuses these activities as the same activity.

Labeled and unlabeled data of 5 subjects From Figure 13 it can be deduced that for the model trained on the labeled and unlabeled data of 5 subjects, the separations of groups is again very visible for *trotting* and *running*. Similarly to the VAE model that was trained on only the labeled data, there is a cluster of *grazing* and *standing* more towards the center of the plot, as can be seen in Figure 14. The activity distribution of these two activities is less visible than the separation of *trotting* and *running*. This is once again very likely because of the nature of the activities. Here, *walking-natural* is also scattered throughout the area of *walking-rider*, like in the model trained on the unlabeled data of 5 subjects.

All unlabeled data The first thing that stands out for the plot of the model trained on all unlabeled data, is that it is located on one line, as can be seen in Figure 13. This could be due to one variable being constant. Comparable to the other plots in Figure 13, the biggest groups are *trotting* and *running* and the other activities are more clustered. The clustering is likely due to the similar nature in these activities, as was explained above. Again, there is a bit of overlap between *walking-natural* and *walking-rider*, for the same reasoning.

All data From Figure 13, it can be seen that *trotting* and *running* are grouped rather well by the model that was trained on all data. It can also be seen that there is a gap in *trotting*, which revels *walking-rider* inside. It is not clearly visible, but is possible that the activities *standing* and *grazing* are hidden within the yellow cluster. As shown in Figure 13, it can be seen that this model looks very similar to the two other models that used labeled data. For this model, where all data is used, the labeled data is still the biggest part. This would explain the similar shape between all models that use the labeled data. These similar shapes suggest that the input data has a big impact on how a model trains. It looks like similar input data results in similar shapes in the latent space representations.

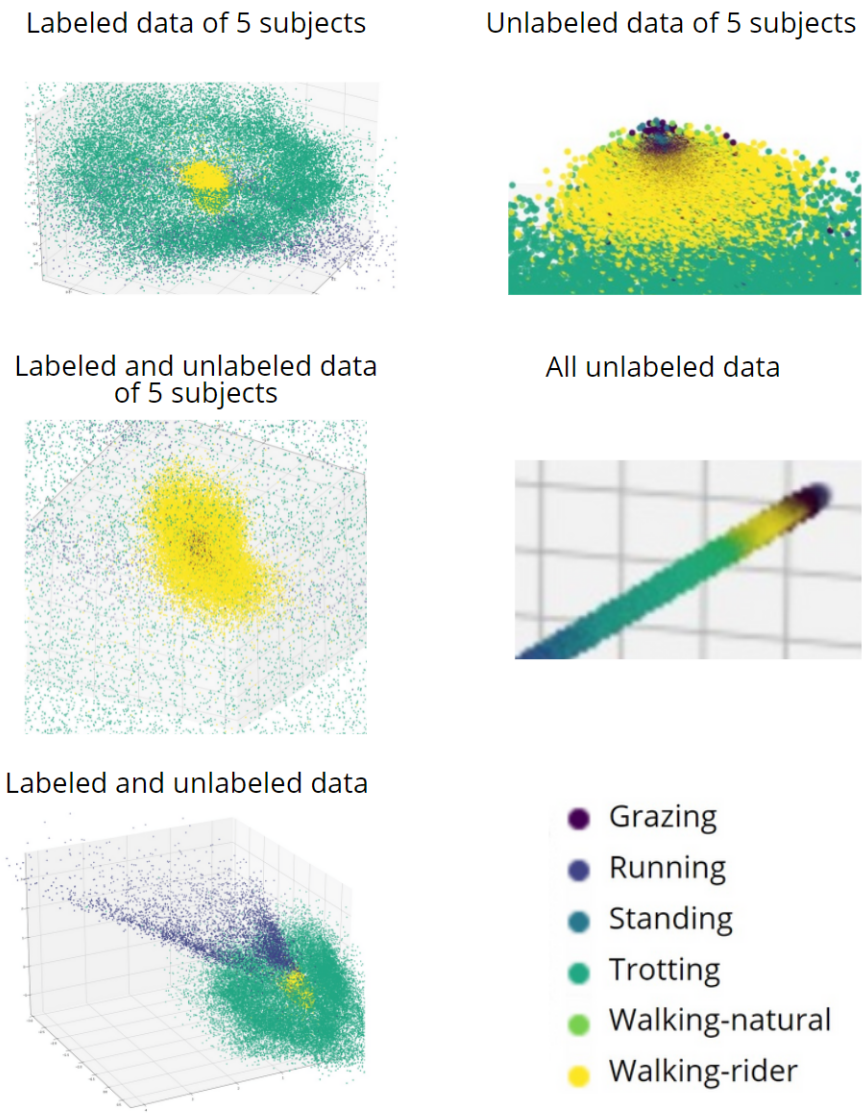

Figure 14: The zoomed in version of the latent representations of the trained VAE models

Similarities and differences between the trained VAE models All models shown in Figure 13 were trained on the same VAE architecture, with the same number of epochs. The only difference in the training process was the input data. This different input data resulted in plots that all managed to separate *trotting* and *running* rather well. This is likely due to the acceleration of these activities. By nature, these activities already differ. The models showed more clustered results for the activities *grazing* and *standing*. This also could be explained by the nature of these activities, since both represent a still-standing horse, so similar accelerations. *Waling-natural*

and *walking-rider* are often confused, again because these activities are very similar by nature. Considering the VAE models were trained on different data, the impact of this can be evaluated. All models trained on the labeled data resulted in similar shapes of the latent representations. Considering the labeled data made up a bigger part of the data, this data was probably dominant in the training process. This would also explain the different shapes from the models trained on only unlabeled data, as these shapes are rather different than the shapes from models trained on the labeled data. That would be because the domination of the labeled data is not present in these models.

Furthermore, it can be seen that the distinction between each activity in Figure 13 is not a straight line, nor a gap. The transitions from activity to activity resemble more of a gradient. Moreover, Figure 13 shows some *trotting* representations in the area where more *running* dots are plotted. This indicates that although there is definitely grouping between the activities, it might not be perfect.

Plotting raw input data Lastly, to investigate the effect of representation learning, the input data is plotted in the same way as the latent representations to compare the results. For this, the x, y and z parameters of the accelerometer sensor values of the labeled data are plotted in a 3D plot, after they have been standardized. The x, y and z values are used, since these make up the A3D vector that is used for this project and this way they can be plotted in a similar manner as the latent representations from the VAE. These values are standardized, since the latent representations also originate from standardized data. This result can be seen in Figure 15. This plot shows that there is no separation between different activities, especially compared to the visible separation that is created by the trained VAE models. The input data is plotted from the five relevant subjects, as well as only one subject, Galoway, to clarify what the data looks like. From b) in Figure 15, it can be seen that most activities are grouped together in one area for a single subject. When plotting all subjects, that group becomes invisible due to outliers.

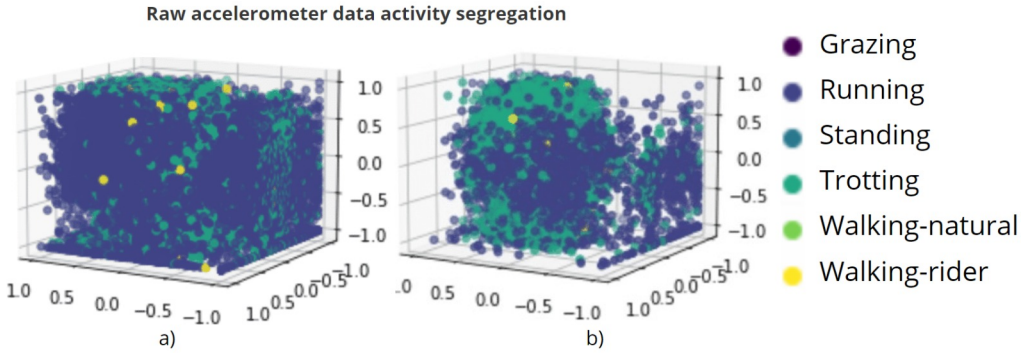

Figure 15: The activity grouping in the raw data of a) 5 subjects and b) 1 subject

When comparing the results of all models to Figure 15, there is quite a big difference. Figure 13 shows obvious grouping between activities and Figure 15 does not. The raw accelerometer data is also much more spread out than the latent representations. Because the latent space representations are clustered together, the plots from Figure 13 look more concise.

Now that the VAE is properly trained, the results of the classifier with different feature extraction methods can be investigated.

6.3 Classification With Statistical Feature Extraction

The classifier is trained on the extracted statistical features of the preprocessed data. The statistical measures used for this are the mean, variance, standard deviation, median, 25^{th} and 75^{th} percentile, maximum, minimum, kurtosis and skewness. The statistical features are extracted for each value in a window. The number of statistical features that were extracted has been varied. The best performing classifier was the one where six features were used. The confusion matrix from Figure 16 shows the results of the classifier using statistical feature extraction with six features.

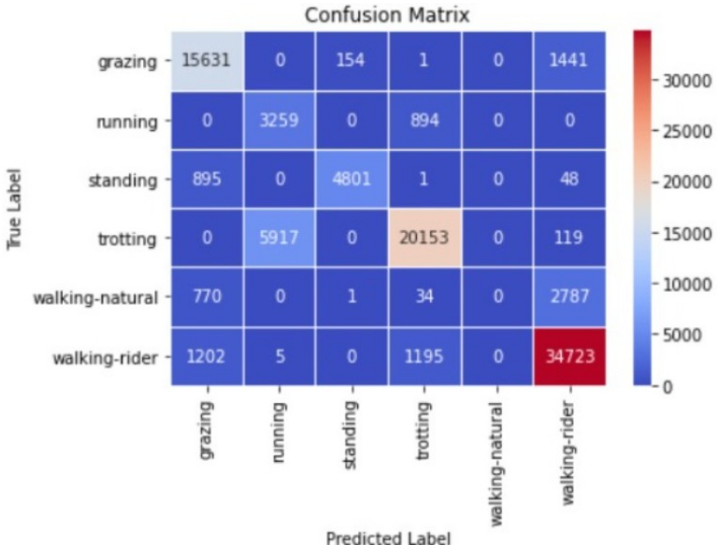

Figure 16: Confusion matrix of classifier with six statistical features; mean, variance, standard deviation, median, 25^{th} and 75^{th} percentile

Feature extraction method	***Accuracy***	***MCC***	***F-score***
Statistical feature extraction	0.86	0.85	0.81

Table 6: Results of training the classifier using six statistical features; mean, variance, standard deviation, median, 25^{th} and 75^{th} percentile

From the confusion matrix it can be deduced that *walking-rider* is classified correctly most often. This is followed by *trotting*, *grazing* and *standing* and *running* respectively. *Walking-natural* has not been classified correctly at all and it is mostly mistaken as *walking-rider*. Another remarkable mistake is that *trotting* and *running* are mixed up more often. Equivalently, *walking-rider* and *grazing* are confused rather frequently as well. *Standing* and *grazing* are mixed up a little as well, although these values are relatively low.

The activities that are confused most are *walking-rider* and *grazing*, as well as *running* and *trotting*. *Walking-rider* and *grazing* are similar activities by nature, in the sense that a horse moves their neck slowly in these activities. Seeing as the sensor is worn around the neck, these activities are likely very similarly reflected in the sensor values. This would explain why both the feature extraction and the classification process have a hard time separating these activities. The same goes for *running* and *trotting*, which by nature are closely related since both require relatively fast acceleration. The accelerometer data is therefore likely very comparable.

These results from the confusion matrix are reflected in the values from Table 6 as well. The accuracy for statistical feature extraction is rather high, and from the confusion matrix it can also be seen that five out of six activities are classified mostly correctly. When looking at the F-score and comparing it to the confusion matrix, it can also be seen that there is only one activity, *running*, which has been predicted falsely more than the true activity. The MCC-score does indicate that the overall prediction is relatively close to the actual labels of the activities.

Lastly, Figure 17 shows the accuracy, MCC-score and F-score for the statistical feature extraction, but this time the number of extracted features is altered. The diagram emphasizes the importance of the input shape for the classifier. As can be deduced from Figure 17, the best results were achieved when six statistical features were fed to the classifier. This gave reasonably higher results than only feeding one statistical feature to the classifier.

The general trend of this graph suggests that a bigger input size results in higher classification performance. This would make sense since this would provide more information to learn from for the classifier. The highest score that is reached for the accuracy and MCC-score, is with six statistical features. For the F-score, the highest value was reached when using four statistical features. Considering the dataset is imbalanced, the F-score gives a good indication of how well the classifier performed. So, following that logic, using four statistical features gives the best classification results. However, F-score and accuracy can be biased, and therefore the MCC-score might indicate better which classification metric is best. This indicates that the use of six statistical features is best.

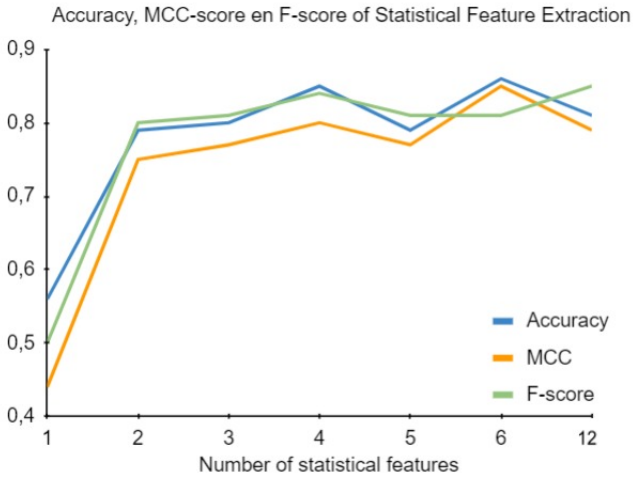

Figure 17: Accuracy, MCC and F-score for a varied number of statistical features. On x-axis: **1**: mean; **2**: mean, variance; **3**: mean, variance, median; **4**: mean, variance, median, standard deviation; **5**: mean, variance, median, standard deviation, 25^{th} percentile; **6**: mean, variance, median, standard deviation, 25^{th} percentile, 75^{th} percentile; **12**: mean, variance, median, standard deviation, 25^{th} percentile, 75^{th} percentile, maximum, minimum, kurtosis, skewness

6.4 Classification With VAE As Representation Learning

The classifier is also trained using the learned representations from a VAE. The representations are extracted from the preprocessed data for each window with a trained VAE model. Several models have been trained, with different types of input data. Figure 18 shows the confusion matrices of the classifier that is trained using these VAE models.

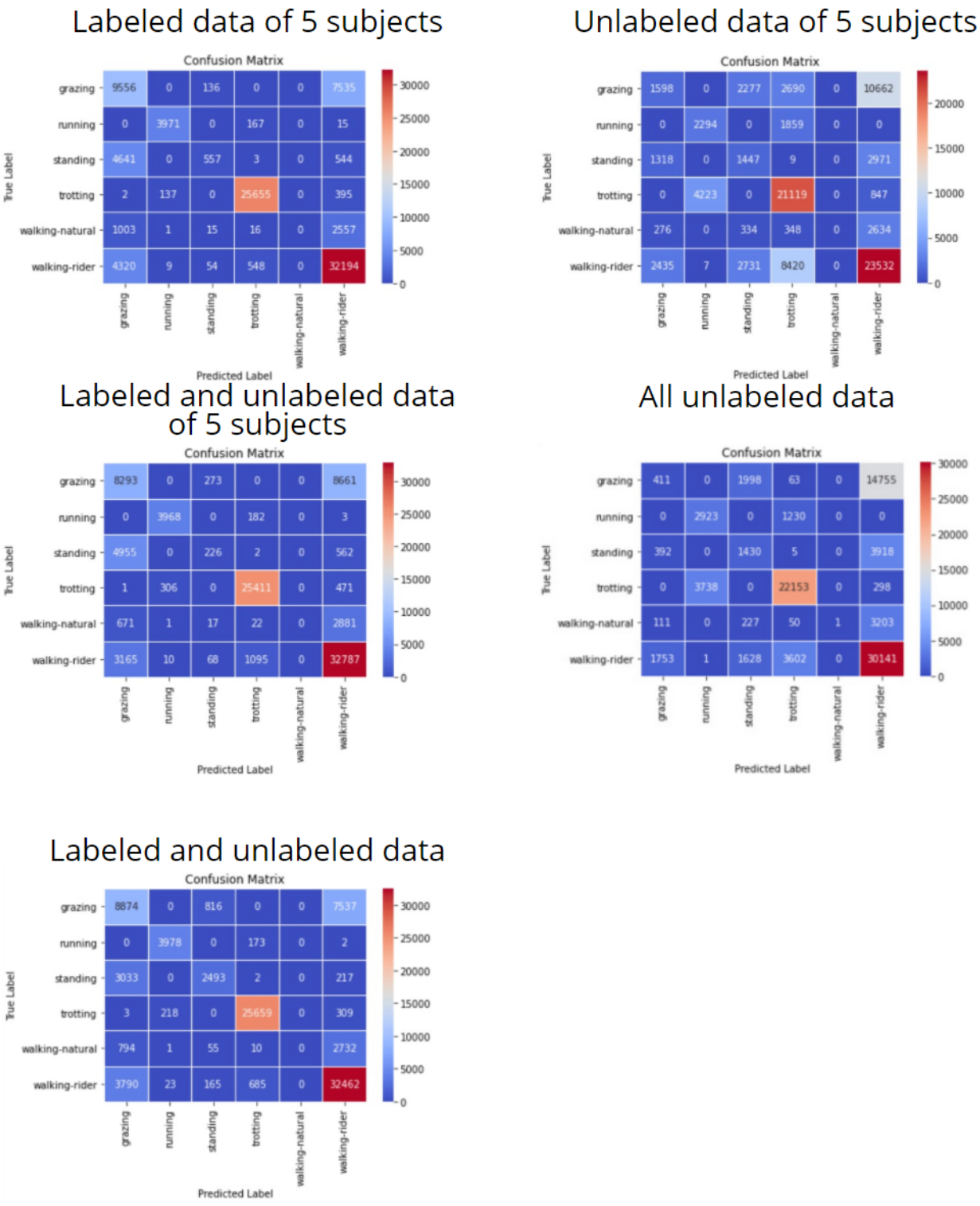

Figure 18: Confusion matrices of classifier using unsupervised representation learning

Feature extraction method	***Accuracy***	***MCC***	***F-score***
Representation learning on labeled data of 5 subjects	0.68	0.78	0.73
Representation learning on unlabeled data of 5 subjects	0.61	0.56	0.50
Representation learning on labeled data and unlabeled data of 5 subjects	0.71	0.75	0.66
Representation learning on all unlabeled data	0.57	0.39	0.46
Representation learning on all data (labeled and unlabeled)	0.70	0.78	0.76

Table 7: Results of training the classifier using unsupervised representation learning

Classification with labeled data of 5 subjects The results from the classifier that uses the VAE that is trained on the labeled data of 5 subjects managed to predict most activities correctly more often than not. Only *walking-natural* is not predicted correctly at all and is mainly confused as *walking-rider*. Also *standing* and *grazing*, as well as *grazing* and *walking-rider* are confused. These confusions are similar to the overlap of these activities that was present for this model in Figure 13. Overall, this model scored an accuracy of 0.68 and a relatively high MCC score; 0.78.

Classification with unlabeled data of 5 subjects This model resulted in a classifier performing relatively bad. As shown in Figure 18, the predictions are rather scattered. Lots of activities are confused: *standing* and *grazing*, *trotting* and *running*, *walking-rider* and *grazing* and lastly *walking-rider* and *standing*. The similar nature of the activities could play a role in this, since *standing* and *running* are not at all confused, which happen to be very different activities by nature. *Trotting* and *running* both have faster acceleration and *walking-rider*, *grazing* and *standing* all are slower activities. Another explanation could be found in the data that is used to train this model. Only the unlabeled data of 5 subjects was used. The classifier is trained on the labeled data of the same 5 subjects. This means there is no overlap in these datasets. It could be that the movements from the unlabeled data contain weird movements that result in representations that confuse the classifier. Another reason could be that the trained VAE model for this classification is trained on the least amount of data. This means that there was little data to learn from and therefore it might not give good representations. Especially not for the data that is used by the classifier. The statistical scores of this classification process from Table 7 are also not very high. This makes sense with the scattered predictions shown in the confusion matrix.

Classification with labeled and unlabeled data of 5 subjects The classifier using the VAE model that was trained on both the labeled and unlabeled data of 5 subjects performed the most accurate out of all classifications using representation learning. The confusion matrix from Figure 18 shows few confusions. Mainly between *grazing* and *walking-rider* and *standing* and *grazing*. However, these confusions are much less common than the other classification processes with representation learning. The accuracy is likely due to the similarity in the data that was used to train the VAE model and the classifier. Both used the labeled data. When looking at Table 7, it can be seen that this classifier performed slightly better than the classifier that used only the labeled data of 5 subjects. The addition of the unlabeled data to the labeled data increased seemed to result in better extracted representations. This could be because more data means more data to learn from for the VAE.

Classification with all unlabeled data The classifier using the VAE model that was trained on all unlabeled data performed the worst. When looking at the confusion matrix in Figure 18, it can be seen that *standing* and *grazing*, *trotting* and *running*, *walking-rider* and *grazing* as well as *walking-rider* and *standing* are confused rather often. Only *walking-rider* and *trotting* are predicted correctly majorly. Especially *grazing* is mistaken as *walking-rider* very often. The similar nature of these activities could explain these results. But a more feasible explanation can be found in the type of data that was used to train the VAE. As described above, the unlabeled data has no correlation to the data that is used to train the classifier. Additionally, this VAE model was trained on data from other horses as well. There are probably too many differences between the two datasets. Lastly, this model showed a very condense latent space representation in 13. This might be too dense and not suitable for the classifier used in this project. The results were reflected in Table 7 as well, since all performance measures scored the lowest for this classification process.

Classification with all data The predictions of the classifier that used all data performed relatively well, as can be deduced from the confusion matrix in Figure 18. Similar to the other classifications, it confused *standing* and *grazing*, as well as *walking-rider* and *grazing*. Overall, it manages to predict all activities accurately most often. From Table 7, it can be seen that this model produced the highest F-score. When looking at the confusion matrix, this can also be seen since overall, most predictions were indeed correct.

Similarities and differences between the classifiers with representation learning Looking at Figure 18 shows that all VAE models results in confusion between *standing* and *grazing*, as well as *walking-rider* and *grazing*. Also, all models resulted in the best predictions for *walking-rider*, followed by *trotting* and *running*. These are also the activities that were most visibly separated in the latent representation plots from Figure 13. It is remarkable that all models did not manage to predict *walking-natural*. Perhaps this is because it is extremely similar to the activity *walking-rider*. Figure 8 also shows that *walking-rider* is much more prominently present in the labeled data than *walking-natural*. It might be that the classification process therefore disregards the *walking-natural* values and sees them as *walking-rider*. Considering *walking-natural* is mostly mistaken for *walking-rider*, as shown in Figure 18, this could be a plausible cause.

6.5 Classification Without Feature Extraction

Finally, the classifier was trained on raw data. The raw data is still preprocessed the same way as for statistical feature extraction and representation learning. However, there were no features extracted this time.

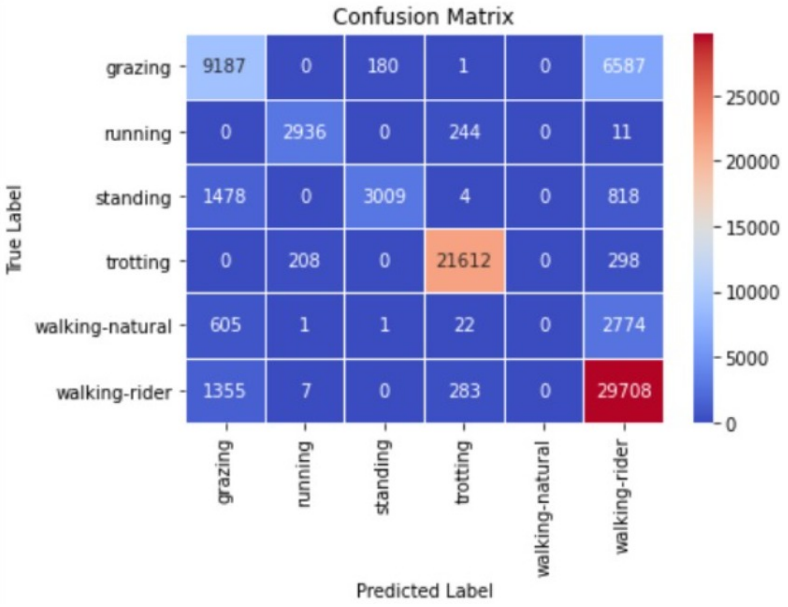

Figure 19: Confusion matrix of classifier using no feature extraction

Feature extraction method	***Accuracy***	***MCC***	***F-score***
No feature extraction	0.81	0.80	0.79

Table 8: Results of training the classifier using no feature extraction

The confusion matrix in Figure 19 shows the results for the classifier that is trained without any feature extraction. From this confusion matrix, it can be seen that the activities *trotting* and *walking-rider* have been predicted the best. This is followed by *grazing*, *standing* and *running*. *Waling-rider* and *grazing* have been mixed up quite a bit, although these activities also have been predicted correctly many times. Similarly, *standing* has been predicted as *grazing* a large number of times. *Walking-natural* has not been predicted correctly once and is mainly classified as *walking-rider*. Overall this classification has been done relatively well.

In Table 8 the classification performance measures also show rather high results. The accuracy is 0.81, the MCC score is 0.80 and the F-score is 0.79. These scores are all relatively high. Only few activities were confused, mainly *standing* and *grazing*, and *walking-rider* and *grazing*. It makes sense that these are confused, since standing and grazing are similar activities by nature, because in both activities the horse is standing still. *Walking-rider* and *grazing* are also confused, again since these activities are similar in nature. Both activities include slow movement of the horses neck. Considering the sensor is worn around the neck, the classifier might see this as similar activities.

6.5.1 Comparing Three Feature Extraction Methods

The discussed results of the model and all feature extraction methods are also compared to each other. The comparison and discussion thereof are below.

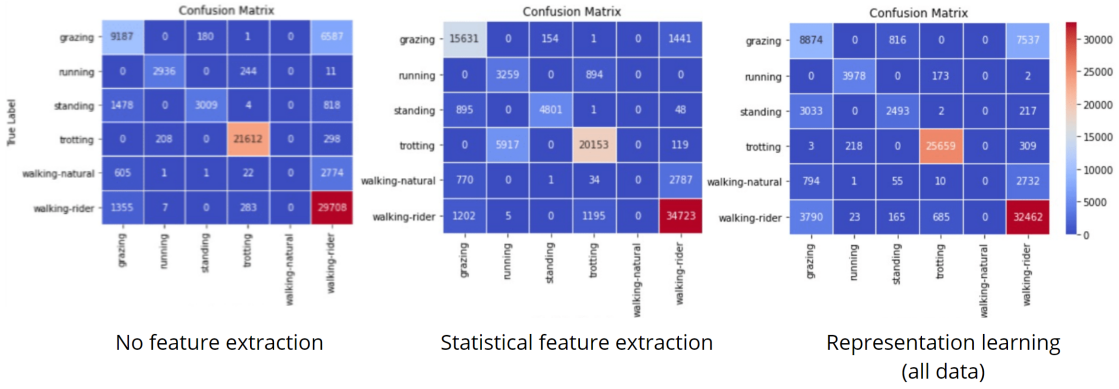

Figure 20: The best confusion matrices from each different feature extraction methods

Figure 20 shows the confusion matrices of all feature extraction methods. The confusion matrix that is chosen for representation learning is chosen is from the best performing classifier with representation learning. These matrices are the same as the ones shown in the results section, but they are placed next to each other for convenience. The first thing that can be deduced from these matrices, is that all feature extraction methods all manage to classify the activities *trotting* and *walking-rider* rather well. The activities *grazing*, *running* and *standing* are followed. All methods also tend to mistake *grazing* and *walking-rider*, which as explained previously are activities that are similar by nature.

Another thing that stands out is that all methods do not, or barely classify *walking-natural* correctly. In all cases it is mostly mistaken for *walking-rider*. This is not surprising because of two reasons. The first one being the similarity in the activities. The second one is due to the imbalance in the dataset. *Walking-rider* is much more prominent in the labeled dataset than *walking-natural*, as was shown in Figure 8. Since the activities are so similar, the data values are likely very similar as well. Because the classifier learns to predict the labels based on sensor values, it groups all sensor values that look like general walking together. It does not manage to differentiate between *walking-natural* and *walking-rider*.

Walking-rider is predicted correctly once, which was by the classifier that used the latent representations. As this activity has been mildly separated from *walking-rider* by the VAE model, as can be seen in Figure 14, it does indicate that the addition representation learning helps the classification process. However, it was only classifier correctly once out of 3592 times. Besides, the separation in Figure 14 is very rare and could even consist of outliers. Therefore, no harsh conclusions can be made from this.

When looking at all confusion matrices, it stands out that the representation learning with unlabeled data, predicts lot more scattered than the statistical- and no feature extraction predictions. Both the accuracy and F-score, and thus the MCC score, are lower, which can also be seen in Figure

21. However, the representation learning classifications using the labeled data provide more similar results to the other feature extraction methods. It can be seen in Figure 21, that all performance measures are higher for both statistical feature extraction and no feature extraction, than that of the representation learning with the VAE. The highest accuracy is scored by the classifier that uses six statistical features. The highest F-score is also achieved by this classifier. However, the highest MCC-score is reached by the classifier that uses 11 statistical features. Overall, the statistical feature extraction method performs best.

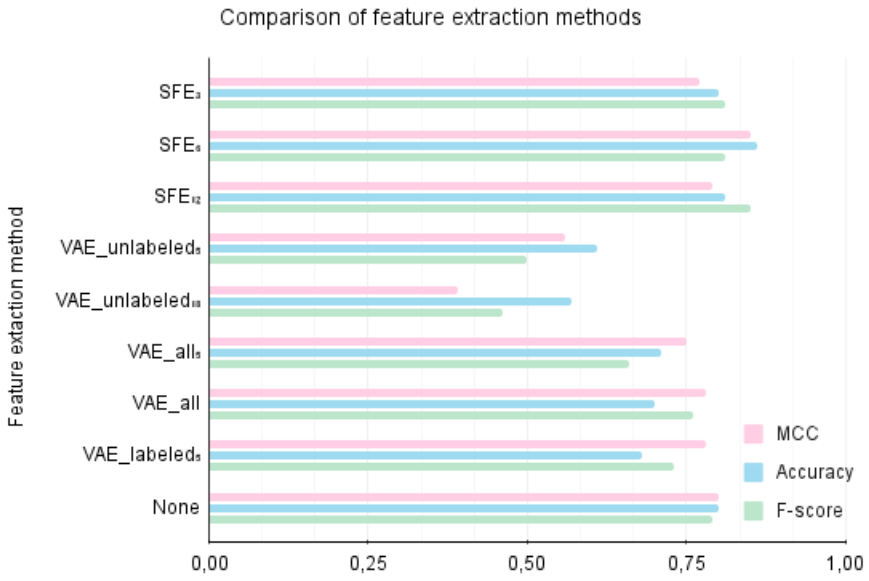

Figure 21: A comparison of the classification performance of all feature extraction methods. **SFE**: Statistical feature extraction with 3 ($\mathbf{SFE}_3$), 6 ($\mathbf{SFE}_6$) and 12 ($\mathbf{SFE}_{12}$) features. $\mathbf{VAE\_unlabeled}_5$: VAE trained on unlabeled data of 5 subjects, $\mathbf{VAE\_unlabeled}_{18}$: VAE trained on unlabeled data of 18 subjects, **VAE_all**: VAE trained on unlabeled and labeled data of 5 subjects, **VAE_all**: VAE trained on labeled data of 5 subjects, $\mathbf{VAE\_labeled}_5$: VAE trained on labeled data of 5 subjects, **None**: without feature extraction

7 Discussion

This section discusses the results that are presented in the results section. Therefore, these results will first be discussed individually and then the addition of a VAE as unsupervised representation learning method to the pipeline is discussed. Additionally, the achieved results will be compared to external relevant research.

7.1 Analysis Of The Trained Variational Autoencoder Models

The trained VAE models were shown in the results section. Their results are discussed below.

In the results the loss of each model was plotted in Figure 12. Ideally, one would want to see an exponential decrease of the loss function while training. But the losses from the trained VAE models here are rather unstable. This could indicate that the loss function was not optimized for the model in combination with the input data. Another possible cause is that the input data did not have enough coherence within itself, that could be learned by the VAE. Both of these affect the model as it does not learn as best as it perhaps could. It therefore also is likely not able to extract the representations as well as it possibly could.

The labeled data used in this project was of a bigger size than the unlabeled data. This is uncommon, since unlabeled data is easier to collect. This is however, because of a loading mistake within ITC Geospatial Computing Portal from the University of Twente. Not all unlabeled data was properly loaded, because of its large size. The loading of the data was cut off, likely because of a memory constraint. This was taken into account and the effects of labeled versus unlabeled data are still valid. It was shown that a bigger input size tends to increase the performance, so perhaps having more unlabeled data would have resulted in better performance.

From the latent representations in the latent representations plot from Figure 13 it could be deduced that the input data is of great importance. It seems to have a big influence on how the model compresses the input data to the latent space representations. The general shape of the latent representations from the models trained using only the labeled data of 5 subjects, the labeled data and the unlabeled data of 5 subjects and all data all looked rather similar. Considering the labeled dataset for these three plots was much bigger than the unlabeled dataset, it would make sense that the labeled data dominates the shape here. However, this is only a suggestion, because when looking at the latent representations shape of the models using only unlabeled data, it can be seen that they look rather different from each other. This is interesting because only a small amount of data is added. The reason for this could be caused by the smaller size of the unlabeled dataset, which might be too small of a dataset for the VAE to give consistent results.

The latent representations of the trained models were compared to the raw input data. However, even though Figure 13 and Figure 15 originate from the same sensor data, they are not perfectly equivalent. So, when this comparison is made, the differences should be disclaimed. Figure 13 consists of the latent representations that are taken from the A3D values of the labeled data from the five relevant subjects. The data points in Figure 15 are taken from the x, y and z parameters of the accelerometer data. This is also from the labeled data from the five relevant subjects. However, the main difference is that the latent representations have already gone through a neural network, and the accelerometer data has not. Nonetheless, they are similar in the sense that both figures do represent the input data for the classifier.

Altogether, the trained VAE models seemed to separate the different activities rather well. Activities that were more similar were grouped together more. The separation was a lot better than the differentiation between activities from the raw data. This segregation of the activities would suggest that when a model is placed within the classification process, the classifier would benefit from the latent representations. This might improve the classification process.

7.2 Classification Using Different Feature Extraction Methods

The classification process has been executed for three different feature extraction methods. The first is statistical feature extraction. The second is representation learning, which uses the VAE models as described above. Lastly, there is no feature extraction. The results that are provided in the Results section are discussed below.

7.2.1 Statistical Feature Extraction

The first classification process used statistical feature extraction. The statistical feature extraction performed best out of all feature extraction methods. Evidently, the statistical features describe the raw data rather well. Because the number of statistical features was varied, the impact of this was shown. Generally, the more statistical features that are extracted, the better the classifier performed. It improves accuracy and the MCC score. However, when 12 statistical features were added, the accuracy and MCC score decreased. This could be caused by the type of statistical features that were added. For example, the maximum and the minimum were computed for each window. These could consist of outliers. If that is the case for all windows, the maximum and minimum would be a relatively constant value. That could confuse the classifier. Another reason could be that the classifier was not compatible with this bigger input size. The vanilla classifier was designed for an input size of `(200,3)`, so an input size of `(200,12)` might be too big.

7.2.2 Representation Learning

The next classification process uses representation learning as feature extraction method in the AAR pipeline. This representation learning is done using several trained VAE models. Below, the implementation of the VAE models will be discussed.

Compatibility of the classifier It is notable that the classification performances using representation learning is lower than the other feature extraction methods. Since it was shown that the learnt representations were better grouped than the raw input data, it suggests that something about the combination of the latent representations and the classifier is not optimal. A classifiers network learns from its input data and predicts classifications that way. As stated before, a bigger input size of a model, provides more data to learn from. The latent representations that were given as an input to the classifier had a size of three, and a classifier using no feature extraction has an input size of one. This makes the input size bigger, more separated per activity, yet still the performance is relatively worse.

There are several possible causes for the poorer performance of the classifier. The first reason would be that the grouping in the latent representation makes the distribution of the data too compressed. The latent space of a VAE compresses the input data to a smaller dimension. This means that the same data has to be presented in a much more compressed form. As was shown from Figures

13 and 15, the range of the data is slightly smaller for the latent representations. The range is even smaller for each activity. Perhaps this range is too small for the classifier to recognize the differences properly.

Another influence on the results of the classifier could be caused by the type of data coming from the latent space. The vanilla classifier that is used for this project is designed for standardized, raw IMU sensor data. The latent representation is data that has already gone through a neural network. This is then fed to another network, the classifier. This makes the classifier's input rather unstable. Since it is nearly impossible to create a perfect model, the output of a model will always contain weaknesses like skewness and over fitting. As was shown in Figure 13, the latent representations are not perfect. There is still some overlap. This is reflected in the data and thus in the classification as well.

Influence of using the labeled data versus the unlabeled data It is also notable that the classifier performed better with VAE models that were trained on labeled data. The reason for this could be that the classifier is trained and tested on this same labeled dataset. The similarities in the classifiers data and the VAE data seem to have a good influence. The classifier looks at its input data and tries to categorize them based on similar values. The VAE does the same, except there is compression and noise generation. When using the same data, or partially the same data, the distribution of both inputs are very similar. The VAE likely compresses the data to representations that correspond to the trained classifier. The classifier using labeled and unlabeled data of five horses supports this, since this performed better than using all the data. The addition of the unlabeled data probably improves the performance because it increases the input size of the VAE, which might improve the learning process. Although the quality of the unlabeled data might not be as good as the labeled data, the fact that the subjects are the same do guarantee similar movements.

Similarly, the classifiers that used VAE models trained on unlabeled data performed the worst. This could be due to the quality and relevance, as well as the size of this data. Firstly, the labeled data has been carefully reviewed and extraordinary errors and outliers are already taken out. It is also sure that the activities in the labeled dataset are only the 18 activities that were shown in Figure 8. From the unlabeled data it is uncertain whether a subject actually stayed within those 18 activities. *Sleeping* is not a labeled activity, for example, but it is possible that the unlabeled dataset contains sleeping data. These values could confuse the classifier.

The last reason for the poor performance with unlabeled data could be because of the size of the data. The unlabeled data was of much smaller size than the labeled data. It might not be enough to properly learn the representations with the smaller amount of input data. Lastly, the unlabeled data.

The effect of using a VAE The results section showed that the addition of a VAE did not improve AAR in the pipeline of this project. Other than the possible causes explained above, this could also be because a VAE is simply not beneficial to the pipeline. A VAE adds variation to its input data. This variation could be misleading for the classifier. Furthermore, for this particular project, an normal autoencoder might be able to learn better from less data, due to having no variation. A normal autoencoder compresses more accurately, since it does not introduce the variation. Therefore, an autoencoder might be more suitable for this project.

Another reason why the VAE might not be perfectly suitable for this project is because it forces a normal distribution in the latent space. Firstly, the vanilla classifier might not expect a normal distribution. This would decrease the performance. Secondly, the normal distribution might not be optimal for the horse dataset used in this project. The dataset contains imbalance and it not normally distributed. Thus, the normal distribution coming from the VAE does not properly represent the dataset, which makes it harder to classify correctly.

All together it can be said that the classifier used in this project might not be the optimal classifier for the latent representations that are used as input for the AAR classification process. The results section showed that the VAE model did manage to separate the animal activities rather well, but this did not result in better classification. The best performances were of those that used the labeled data, which is similar to the training and testing data of the classifier. There are several reasons why the classifier might not be suited for the classification process that is used in this project's particular pipeline. It is because the range and type of the data is different than the IMU sensor data, which the vanilla classifier is built for. The VAE might also not be a suitable addition to the pipeline, since it introduces variation and forces a normal distribution.

7.2.3 No Feature Extraction

The last feature extraction method that is tested for with the AAR pipeline uses no feature extraction.

The most important aspect about this method, is that the input data of the classifier is actually meant for the classifier used in the project. the classifier is designed for accelerometer data. This is a huge benefit, since the learning process is more efficient this way.

Although some activities are confused, the classification of the raw, preprocessed data is done rather well. These results are likely due to the data that is suitable for the classifier. The activities that are confused are by nature very similar activities.

7.3 Comparison To External Research

The nature of this research if very similar to the research that has been done by Kamminga et al. [6]. That research uses the same data and also uses statistical feature extraction and representation learning as an addition to an AAR pipeline.

The statistical feature extraction results can be compared to statistical feature extraction that was performed on the same dataset, but with using a different classifier. This method from Kamminga et al. [6] produced the results that can be seen in Figure 22. Since the labeled data that is used for this project is most similar to 100% of the data from Kamminga et al. [6], the comparison will be made to the far-right values of Figure 22. The statistical feature extraction methods done by Kamminga et al. uses time- and frequency domain measures, as well as a combined version of the two. The combined version produced the highest F1-score, followed by the time domain and then the frequency domain. This is also the order of the number of features, so perhaps that could play a part in these results.

The statistical features that are extracted in this project are most similar to the features that are extracted in the time domain, so only that value will be compared to this project. From Figure 22, it can be deduced that the time domain extracted features resulted in an F-score of 93%. This is higher

than the F-score of this project. There are two possible reasons for this. The first one is that the input size of this classification process is slightly bigger. Kamminga et al. [6] used twelve statistical features as an input and this project used six. Figure 17 indicated that more statistical features result in better performances. Therefore, the higher F-score could be explained by the input size. The second reason for the higher F-score could be caused by the use of a different classifier. Each classifier has a different network and thus a different way of learning to classify. Some classifiers fit different types of data better than others. Perhaps the classifier used by Kamminga et al. [6] was more appropriate for the statistical features than the classifier used in this project.

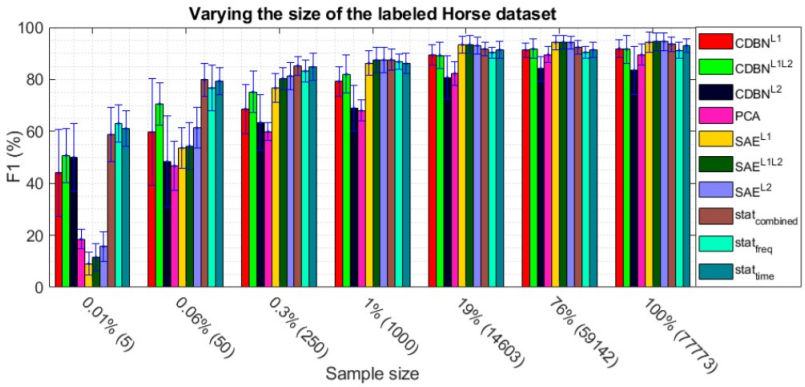

Figure 22: AAR performances using different feature extraction methods by Kamminga et al. [6]

In the research by Kamminga et al. [6], more feature extraction methods were used. One of them is a sparse autoencoder for representation learning. For this, the approach was the same, but the exact execution of the pipeline differs to this project, as the pipeline by Kamminga et al. uses a Support Vector Machine classifier, instead of a vanilla classifier. As stated by Kamminga et al., a Support Vector Machine is more robust against the higher dimensionality of the input data.

Since the results of Kamminga et al. suggested improvement of the classifications by adding more generative models, it is insightful to compare the results to the pipeline with a VAE, which is a more generative model than a sparse autoencoder. Figure 23 [6] shows the F-scores for sparse autoencoders (SAE), convolutional deep belief networks (CDBN) and principal component analysis (PCA). To compare this to the results of the pipeline from this project with the VAE, only the AAR performances that use 100% of the unlabeled data will be looked at. The F-score from Kamminga et al. when using a sparse auto encoder for AAR, is 0,8086. The F-score from the VAE of this project reached at most a value of 0.76. This is not that big of a difference to the results from Kamminga et al., but it did not improve the classification process. The only differences are in the representation learning model, and in the classifier model.

sample size	0.3% (250)		1% (1000)		19% (14553)		76% (58935)		100% (77523)	
representation	F_1	σ	F_1	σ	F_1	σ	F_1	σ	F_1	σ
$CDBN^{L1}$	65,17	4,40	66,78	5,71	72,14	5,11	72,62	5,69	70,01	6,86
$CDBN^{L1L2}$	76,45	6,81	76,16	5,90	75,30	6,12	76,87	5,48	78,00	7,90
$CDBN^{L2}$	60,59	10,83	61,15	10,20	62,43	7,64	63,24	9,98	65,79	9,57
PCA	59,18	4,78	59,13	4,48	58,96	3,62	58,99	3,68	56,30	5,06
SAE^{L1}	78,47	4,83	77,63	5,71	75,14	4,53	76,39	6,70	74,19	5,75
SAE^{L1L2}	**80,01**	5,09	**78,74**	5,56	76,40	4,91	80,05	4,68	79,77	6,90
SAE^{L2}	76,45	4,70	76,04	4,71	**77,26**	4,63	**81,50**	5,24	**80,86**	9,71

Figure 23: AAR performance using different sizes of unlabeled data.

Compared to a sparse autoencoder, a VAE introduces noise to generate new data. This could be a cause of the poorer performance in this project. The added feature extraction of an autoencoder could be a good addition to the pipeline but the noise generation of the VAE might be too confusing for the classifier. Another reasoning could be that the classifier of Kamminga et al. is more suited for the input from the sparse autoencoder.

7.4 The Addition Of Unsupervised Representation Learning To AAR Pipeline

The feature extraction methods have been discussed and compared to external research. From this, the advantages and disadvantages of adding a VAE as unsupervised representation learning method can be deduced. The list of disadvantages below is longer than the list of advantages. This implies that the addition of a VAE to the AAR pipeline does not improve the classification. Because the statistical feature extraction showed to improve the performance, the poorer performance is likely not because of the addition of feature extraction in general. The main reason is more likely the compatibility of the VAE to the classifier. The friction the VAE introduces between the two models could be avoided by using a normal autoencoder or another generative model. Another solution would be to change the classifier.

Advantages

- The VAE grouped the input data decently, especially compared to the raw input data.
- Although not from the VAE, the addition of representations can be beneficial.

Disadvantages

- The model was unstable which does not guarantee proper input data for the classifier.
- A VAE compresses the data too much for the vanilla classifier.
- It changes the type of data from IMU sensor data to representations. This could be too unstable.

- A VAE adds variation to its input data, which can confuse the classifier.
- A VAE forces a normal distribution which might not be the best for the horse dataset.

7.5 Summary

From the discussed results several things can be concluded. First of all, the VAE model managed to separate activities rather well. Especially activities that are very different by nature were easier to separate. More similar activities were mixed up more often. These same results were found in the classification process. Although, for all feature extraction methods, the results were affected by the imbalance of the dataset.

From the different feature extraction methods, the statistical feature extraction performed best. This is followed by no feature extraction and lastly by the representation learning. Generally speaking, the more statistical features that were extracted, the better the performance. Although there is a limit to this. Additionally, a VAE trained on unlabeled data performed worse than a VAE model that was trained on labeled data of the five subjects. This is likely because with the labeled data, the model is more tailored for the AAR pipeline, which uses the same data. The poorer performance of the additional representation learning is likely due to the unsuitability of the data to the classifier and the varaition and normal distribution that is enforced by the VAE.

Because the addition of feature extraction did improve the performance, one could say that another type of feature extraction, or even representation learning, can improve the performance. Feature extraction is a valuable addition, but the research from this project suggests that a VAE is not the best method for this.

Comparing the results to the research by Kamminga et al. [6] shows that the VAE performs a little worse than a sparse autoencoder. This is likely because of the different classifier that is used by Kamminga et al., which might be more suitable than the vanilla classifier of this project.

8 Conclusion

"To what extent does a VAE result in a better classification for animal activity recognition of raw IMU animal activity data?" That was the research question of this project. To answer this, several steps have been taken. Literature research has been done to find out why VAE promises better results. Then, an approach has been created to evaluate this. This approach consisted of testing statistical feature extraction, representation learning with a VAE and no feature extraction method. A pipeline was created, a VAE model was trained and the results were displayed and discussed.

The results showed that the best performing AAR pipeline was the one using statistical feature extraction. This was followed by the normal AAR pipeline and lastly the pipeline that used unsupervised representation learning. The more statistical features that were presented to the classifier as input data, the better the performance got, although there was a limit to this. The results also showed that the used VAE model for the representation learning pipeline, was trained rather well. It was able to segregate different animal activities into groups. It was better at differentiating vastly different activities, than activities that are more similar.

The results from each AAR pipeline also reflected an easier separation between drastically different activities than more similar activities. It was also shown that the imbalance of the dataset influenced each pipeline, mainly for the activity *walking-natural*.

Considering the statistical feature extraction performed better than the normal AAR pipeline, it can be implied that feature extraction can be a beneficial addition to AAR. The unsupervised representation learning method did not perform better than a normal classifier or statistical feature extraction, which might be due to the compatibility of the VAE to the classifier. The compression, variation and normal distribution that is added to the raw input data by the VAE might not be optimal for the classifier.

Comparing the results to the research by Kamminga et al. [6] shows that the VAE performs slightly worse than a sparse autoencoder. This is likely because of the different classifier that is used by Kamminga et al., which might be more suitable than the vanilla classifier of this project. It could also be due to the noise generation in a VAE that is not present in a sparse autoencoder.

To conclude and to answer the research question, two things can be said. First of all, statistical feature extraction showed to improve the classification results. This suggests that any sort of feature extraction can potentially be beneficial to the AAR pipeline. Additionally, the latent representations of the VAE managed to represent the different activities in segregated manner. However, the VAE did not improve he performance of the AAR pipeline and even performed less good than the AAR pipeline without any feature extraction. However, it did score decent when using the same data to train the VAE as the data to train the classifier. Considering the well segregated activities, it is likely due to the compatibility of the VAE and the classifier.

8.1 Recommendations

Because the unlabeled dataset that was used was of a large size, there were flaws in the loading of this data. The loading got cut off, which resulted in more labeled than unlabeled data being used. This was taken into account within the scope of this project, but perhaps more unlabeled data would result in better performance of the AAR pipeline.

Additionally, it would be beneficial to further investigate the imbalance of the dataset. The results of this project were affected by this imbalance and it would be insightful to evaluate the same process without the imbalance. The activities that were more prominent in the dataset showed dominance in the results. The lesser present activities suffered from this. Leveling the presence of the activities in the dataset could be a possible solution for this.

8.2 Future Work

The main problem that was found in the representation learning, is that the classifier did not seem to be compatible with the latent representations. This data is more compressed and has different values than the sensor data, which the vanilla classifier was built for. The research from Kamminga et al. [6] performed slightly better on the same dataset, but a different classifier was used. Although this is not the only reason for the difference in performance, it indicates that a different classifier could be a solution for even better performances. To further investigate the addition of a VAE, the same pipeline should be tested with a classifier that is more suitable for latent representations as input data.

Additionally, the noise that was introduced by the VAE could be omitted by using a normal autoencoder. This does not introduce variation and does not enforce a normal distribution. Therefore, the addition of a normal autoencoder as unsupervised representation learning method could be further investigated.

9 References

[1] B. D. Scott and M. Martin, "Understanding Vital Life Signs in Horses," pp. 1–5, 2014.

[2] DeepAI, *Feature Extraction*. [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/feature-extraction#:~:text=Feature%20extraction%20is%20a%20process,of%20computing%20resources%20to%20process..

[3] Y. Bengio, A. Courville, and P. Vincent, "Representation Learning: A Review and New Perspectives," Jun. 2012. [Online]. Available: http://arxiv.org/abs/1206.5538.

[4] E. V. Añazco, P. R. Lopez, H. Park, N. Park, and T. S. Kim, "Human activities recognition with a single writs imu via a variational autoencoder and android deep recurrent neural nets," *Computer Science and Information Systems*, vol. 17, no. 2, pp. 581–597, Jun. 2020, ISSN: 24061018. DOI: 10.2298/CSIS190920005V.

[5] L. Bai, C. Yeung, C. Efstratiou, and M. Chikomo, "Motion2Vector: Unsupervised learning in human activity recognition using wrist-sensing data," in *UbiComp/ISWC 2019- - Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers*, Association for Computing Machinery, Inc, Sep. 2019, pp. 537–542, ISBN: 9781450368698. DOI: 10.1145/3341162.3349335.

[6] J. W. Kamminga, D. V. Le, and P. J. M. Havinga, "Towards deep unsupervised representation learning from accelerometer time series for animal activity recognition," *MileTS '20*, 2020.

[7] Y. Li, D. Shi, B. Ding, and D. Liu, "LNAI 8891 - Unsupervised Feature Learning for Human Activity Recognition Using Smartphone Sensors," Tech. Rep.

[8] Odongo Steven Eyobu and Dong Seog Han, "Feature Representation and Data Augmentation for Human Activity Classification Based on Wearable IMU Sensor Data Using a Deep LSTM Neural Network," *Sensors*, vol. 18, no. 2892, pp. 5–23, 2018. [Online]. Available: chrome-extension://dagcmkpagjlhakfdhnbomgmjdpkdklff/enhanced-reader.html?pdf=https%3A%2F%2Fbrxt.mendeley.com%2Fdocument%2Fcontent%2F460c6a51-eb02-3328-9958-c8bd8c1b3993.

[9] M. S. Siraj and M. A. Ahad, "A Hybrid Deep Learning Framework using CNN and GRU-based RNN for Recognition of Pairwise Similar Activities," in *2020 Joint 9th International Conference on Informatics, Electronics and Vision and 2020 4th International Conference on Imaging, Vision and Pattern Recognition, ICIEV and icIVPR 2020*, Institute of Electrical and Electronics Engineers Inc., Aug. 2020, ISBN: 9781728193311. DOI: 10.1109/ICIEVicIVPR48672.2020.9306630.

[10] D. Žagar and r. i. i. t. O. Sveučilište Josipa Jurja Strossmayera u Osijeku Fakultet elektrotehnike, *On the Benefits of Deep Convolutional Neural Networks on Animal Activity Recognition*. 2020, ISBN: 9781728197593. [Online]. Available: https://ieeexplore-ieee-org.ezproxy2.utwente.nl/stamp/stamp.jsp?tp=&arnumber=9263702.

[11] J. W. Kamminga, N. Meratnia, and P. J. Havinga, "Dataset: Horse movement data and analysis of its potential for activity recognition," in *DATA 2019 - Proceedings of the 2nd ACM Workshop on Data Acquisition To Analysis, Part of SenSys 2019*, Association for Computing Machinery, Inc, Nov. 2019, pp. 22–25, ISBN: 9781450369930. DOI: 10.1145/3359427.3361908.

[12] E. Casella, A. R. Khamesi, and S. Silvestri, "Smartwatch application for horse gaits activity recognition," in *Proceedings - 2019 IEEE International Conference on Smart Computing, SMARTCOMP 2019*, Institute of Electrical and Electronics Engineers Inc., Jun. 2019, pp. 409–416, ISBN: 9781728116891. DOI: 10.1109/SMARTCOMP.2019.00080.

[13] G. Demir and A. T. Erman, "Activity recognition and tracking system for domestic animals," in *26th IEEE Signal Processing and Communications Applications Conference, SIU 2018*, Institute of Electrical and Electronics Engineers Inc., Jul. 2018, pp. 1–4, ISBN: 9781538615010. DOI: 10.1109/SIU.2018.8404784.

[14] S. Sonawani and D. Mukhopadhyay, "A Decision Tree Approach to Classify Web Services using Quality Parameters," Tech. Rep.

[15] International Association for Pattern Recognition, Zhongguo ke xue yuan, and Chinese Association of Automation, *2018 24th International Conference on Pattern Recognition (ICPR)*. 2018, pp. 1402–1407, ISBN: 9781538637883.

[16] M. Shoaib, S. Bosch, O. Incel, H. Scholten, and P. Havinga, "A Survey of Online Activity Recognition Using Mobile Phones," *Sensors*, vol. 15, no. 1, pp. 2059–2085, 2015, ISSN: 1424-8220. DOI: 10.3390/s150102059. [Online]. Available: http://www.mdpi.com/1424-8220/15/1/2059/.

[17] M. Phi, *Illustrated Guide to LSTM's and GRU's: A step by step explanation*, 2018.

[18] M. Stewart, *Guide to Classification on Imbalanced Datasets*, May 2021.

[19] D. Chicco and G. Jurman, "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation," *BMC Genomics*, vol. 21, no. 1, Jan. 2020, ISSN: 14712164. DOI: 10.1186/s12864-019-6413-7.

[20] J. W. Kamminga, L. M. Janßen, N. Meratnia, and P. J. M. Havinga, "Horsing Around-A Dataset Comprising Horse Movement," 2019. DOI: 10.4121/uuid:2e08745c-4178-4183-8551-f248c992cb14. [Online]. Available: https://doi.org/10.4121/uuid:2e08745c-4178-4183-8551-f248c992cb14..

[21] "Dataset1,"

[22] R. Arablouei, L. Currie, B. Kusy, A. Ingham, P. L. Greenwood, and G. Bishop-Hurley, "In-situ classification of cattle behavior using accelerometry data," *Computers and Electronics in Agriculture*, vol. 183, p. 106 045, Apr. 2021. DOI: 10.1016/j.compag.2021.106045.

[23] *Geospatial Computing Portal*, 2020. [Online]. Available: https://crib.utwente.nl/.

[24] *Project Jupyter*, Apr. 2021. [Online]. Available: https://jupyter.org/.

[25] *pandas - Python Data Analysis Library*, Apr. 2021. [Online]. Available: https://pandas.pydata.org/.

[26] *NumPy*, 2020. [Online]. Available: https://numpy.org/.

[27] *PyTorch*, Apr. 2021. [Online]. Available: https://pytorch.org/.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[29] *Keras: the Python deep learning API*. [Online]. Available: https://keras.io/.

[30] C. Leifer, *peewee*. [Online]. Available: http://docs.peewee-orm.com/.

[31] M. Waskom, *seaborn: statistical data visualization*, 2020. [Online]. Available: https://seaborn.pydata.org/.

10 Appendix

10.1 The Pipeline

```
# ------------------------------------------------------ #
#  Imports:
# ------------------------------------------------------ #
import numpy as np
import pandas as pd
import glob
import torch
import os

from sklearn import preprocessing
from scipy import stats, signal
from scipy.stats import norm, kurtosis, skew
from keras.utils import np_utils
from sklearn.utils import shuffle
from tqdm import tqdm
from torch import nn, optim
from torch.nn import functional as F


# ------------------------------------------------------ #
#  Variational Autoencoder:
# ------------------------------------------------------ #

INPUT_SIZE = 200
LAYER_SIZE = 100
LATENT_SIZE = 3


class VAE(nn.Module):
    def __init__(self):
        super(VAE, self).__init__()

        # Encoder
        self.l1 = nn.Linear(INPUT_SIZE, LAYER_SIZE)
        self.l12 = nn.Linear(LAYER_SIZE, LAYER_SIZE//2)
        self.l13 = nn.Linear(LAYER_SIZE//2, LAYER_SIZE//4)
        self.l3a = nn.Linear(LAYER_SIZE//4, LATENT_SIZE)
        self.l3b = nn.Linear(LAYER_SIZE//4, LATENT_SIZE)
        # Decoder
        self.l4 = nn.Linear(LATENT_SIZE, LAYER_SIZE//4)
        self.l5 = nn.Linear(LAYER_SIZE//4, LAYER_SIZE//2)
        self.l6 = nn.Linear(LAYER_SIZE//2, LAYER_SIZE)
        self.l7 = nn.Linear(LAYER_SIZE, INPUT_SIZE)
```

```
        self.record = False
        self.records = []

    def set_record(self, boolean):
        self.record = boolean

    def get_records(self):
        return self.records

    def reset_records(self):
        self.records = []

    def encode(self, x):
        out = F.relu(self.l1(x))
        out = F.relu(self.l12(out))
        out = F.relu(self.l13(out))
        return self.l3a(out), self.l3b(out)

    # Taken from https://github.com/pytorch/examples/blob/master/vae/main.py
    def reparameterize(self, mu, sigma):
        std = torch.exp(0.5*sigma)
        eps = torch.randn_like(std)

        res = mu + eps*std
        res = torch.clamp(res, min=-5.0, max=5.0)
        if self.record:
            self.records.append(res)
        return res

    def decode(self, x):
        out = F.relu(self.l4(x))
        out = F.relu(self.l5(out))
        out = F.relu(self.l6(out))
        return torch.sigmoid(self.l7(out))

    def forward(self, x):
        mu, sigma = self.encode(x)
        z = self.reparameterize(mu, sigma)
        return self.decode(z), mu, sigma


def model_train(data):
    EPOCHS = 50
    model = VAE()
    optimizer = optim.Adam(model.parameters(), lr=3e-4)
```

```
    record_loss = []

    def loss_function(recon_x, x, mu, sigma):
        BCE = F.binary_cross_entropy(recon_x, x, reduction='sum')
        KLD = torch.sum(1 + sigma - mu.pow(2) - sigma.exp())
        return BCE

    model.train()
    for w in tqdm(range(EPOCHS)):
        t_loss = 0
        for d in data:
            model.zero_grad()
            recons, mu, sigma = model(torch.unsqueeze(d, 0))
            loss = loss_function(recons, torch.unsqueeze(d, 0), mu, sigma)
            record_loss.append(loss)
            loss.backward()
            optimizer.step()

    return model, record_loss


# Function to train VAE
def model_train(data):
    EPOCHS = 3
    model = VAE()
    optimizer = optim.Adam(model.parameters(), lr=3e-4)

    def loss_function(recon_x, x, mu, sigma):
        BCE = F.binary_cross_entropy(recon_x, x, reduction='sum')
        return BCE

    model.train()
    for w in tqdm(range(EPOCHS)):
        t_loss = 0
        for d in data:
            model.zero_grad()

            recons, mu, sigma = model(torch.unsqueeze(d, 0))

            loss = loss_function(recons, torch.unsqueeze(d, 0), mu, sigma)
#             t_loss += loss
            loss.backward()

            optimizer.step()

#         print(f"Total loss VAE for epoch {w}: {t_loss}")
```

```
    return model

# ------------------------------------------------- #
#  Pre-Processing Constants:
# ------------------------------------------------- #

# Label encoder used to get a numeric representation of a label
le = preprocessing.LabelEncoder()

# The activities
LABELS = ['grazing', 'running',  'standing', 'trotting', 'walking-natural', 'walking-ri
print(len(LABELS))

# Add columns to drop from dataframe
REMOVE_COLUMNS = ['Ax', 'Ay', 'Az', 'Gx', 'Gy', 'Gz', 'Mx', 'My', 'Mz', 'G3D', 'M3D']

# Add subjects you want to include
SUBJECTS = ['Zafir', 'Driekus', 'Patron', 'Galoway', 'Happy']
# SUBJECTS = ['Galoway']

# Amount of features (xyz acc / xyz gyr)
N_FEATURES = 3

# Name of the column used as output
OUTPUT_LABEL = 'ActivityEncoded'

# Sliding windows parameters
TIME_PERIODS = 200
STEP_DISTANCE = 100

# Datasets
PATH = 'Data/*'
FILES = sorted(glob.glob(PATH))

# Loading the VAE model
MODEL = torch.load("alldata.bin").cpu()

# ------------------------------------------------- #
#  Helper functions:
# ------------------------------------------------- #


def create_dataframe(files):
    """
    Simple function to set up dataframe and initial clean-up of the data
    files: path to files
```

```
    returns: dataframe
    """
    result = pd.DataFrame()
    # Pick only the files in SUBJECTS
    matching = [f for f in files if any(s in f for s in SUBJECTS)]

    for file in tqdm(matching):
        csv = pd.read_csv(file)
        csv['filename'] = file
        result = result.append(csv)

    # remove redundant columns
    result.drop(REMOVE_COLUMNS, axis=1, inplace=True)
    result = select_activities(result)
    # create a new column with a unique integer value for each label
    result[OUTPUT_LABEL] = le.fit_transform(result['label'].values.ravel())
    return result


def select_activities(df):
    df['label'] = df['label'].replace(to_replace=['trotting-natural'], value='trotting'
    df['label'] = df['label'].replace(to_replace=['trotting-rider'], value='trotting')
    df['label'] = df['label'].replace(to_replace=['running-natural'], value='running')
    df['label'] = df['label'].replace(to_replace=['running-rider'], value='running')
    result = df[df['label'].isin(LABELS)]
    return result


# Butterworth filter
def filter(df):
    sos = signal.butter(N=3, Wn=30, btype='lowpass', fs=100, output='sos')
    df['A3D'] = signal.sosfilt(sos, df['A3D'])
    return df


#  Creates train and test set based on subjects
def split_by_subject(df, name):
    test = df[df['filename'].str.contains(name)]
    train = df[~df['filename'].str.contains(name)]
    return train, test


# Scaling features to (0,1) scale
def feature_scaling(df):
    train_x_max = df['A3D'].max()
    pd.options.mode.chained_assignment = None
```

```
    # divide all 3 axis with the max value in the training set
    df['A3D'] = df['A3D'] / train_x_max

    return df


# Creates windows
def create_windows(df, time_steps, step, label_name):
    windows = []
    labels = []
    for i in range(0, len(df) - time_steps, step):
        a3d = df['A3D'].values[i: i + time_steps]
        # Retrieve the most often used label in this segment
        label = stats.mode(df[label_name][i: i + time_steps])[0][0]
        windows.append([a3d])
        labels.append(label)
    # Bring the segments into a better shape
    reshaped_windows = np.asarray(windows, dtype=np.float32).reshape(-1, time_steps, 1)
    labels = np.asarray(labels)

    return reshaped_windows, labels


# Gets statistical features
def get_stat_features(data):
    res = []
    for i, window in tqdm(enumerate(data)):
        maxi = np.amax(window)
        mini = np.amin(window)
        mean = np.mean(window, axis = None)
        std = np.std(window, axis = None)
        median = np.median(window, axis = None)
        cov = math.sqrt(std)
        twf_perc = np.percentile(window, 25, axis = None)
        svf_perc = np.percentile(window, 75, axis = None)
        skewness = skew(window)
        kurt = kurtosis(window)
        stats = np.array([mean, std, median,
        cov, twf_perc, svf_perc,
        maxi, mini, skewness, kurt, cov])
        res.append(stats)
        res[i] = np.full((200, 11), stats)
    return np.asarray(res)
```

```
# Puts window through VAE model and gets latent representations
def get_latent_representations(data, w=True):
    nr_of_windows = data.shape[0]
    data = data.reshape(nr_of_windows, 200)
    tensor = torch.tensor(data)
    trained = []
    for i, window in tqdm(enumerate(tensor)):
        reps = MODEL(torch.unsqueeze(window, 0))
        reps = torch.detach(reps[1])
        trained.append(reps.numpy())
        trained[i]= np.full((200,3), reps)
    trained = np.asarray(trained)
    return trained


# Reshape input into a format compatible with the NN
def reshape_input(x, shape):
    result = x.reshape(x.shape[0], shape)
    return result


# Apply one hot coding to output
def encode_output(y, classes):
    result = np_utils.to_categorical(y, classes)
    return result


def preprocess_training(df, input_shape, num_classes):
    train = feature_scaling(df)
    x_train, y_train = create_windows(train, TIME_PERIODS, STEP_DISTANCE, OUTPUT_LABEL)
    x_train, y_train = shuffle(np.array(x_train), np.array(y_train))
#     x_train = get_latent_representations(x_train)
#     x_train = get_stat_features(x_train)
    x_train = reshape_input(x_train, input_shape)
    x_train = x_train.astype('float32')
    y_train = y_train.astype('float32')
    y_train = encode_output(y_train, num_classes)

    return x_train, y_train


def preprocess_test(df, input_shape, num_classes):
    test = feature_scaling(df)
    x_test, y_test = create_windows(test, TIME_PERIODS, STEP_DISTANCE, OUTPUT_LABEL)
    x_test = get_latent_representations(x_test)
#     x_test = reshape_input(x_test, input_shape)
```

```
#     x_test = get_stat_features(x_train)
    x_test = x_test.astype('float32')
    y_test = y_test.astype('float32')
    y_test = encode_output(y_test, num_classes)

    return x_test, y_test


def preprocess(df, test_subject):
    input_shape = (TIME_PERIODS * N_FEATURES)
    num_classes = len(LABELS)
    df = filter(df)
    train, test = split_by_subject(df, test_subject)
    x_train, y_train = preprocess_training(train, input_shape, num_classes)
    x_test, y_test = preprocess_test(test, input_shape, num_classes)

    return x_train, y_train, x_test, y_test, input_shape, num_classes

# ------------------------------------------ #
#  Classifier:
# ------------------------------------------ #

import importlib
import ipynb.fs.full.Database as db
db = importlib.reload(db)

import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Reshape
from sklearn import metrics
from matplotlib import pyplot as plt
import seaborn as sns
import uuid

import glob
import numpy as np

# Hyper-parameters
BATCH_SIZE = 512
EPOCHS = 25

# Datasets
PATH = 'Data/*'
FILES = sorted(glob.glob(PATH))

# Set to false to disable database support
```

```
db_enabled = True

# Include the model description here, as this is NOT included in the parameters
DESCRIPTION = "Neural Network with 6 layers (Reshape, Dense x3, Flatten, Dense)"
PARAMETERS = "TIME_PERIOD: " + str(TIME_PERIODS) +
", STEP_DISTANCE: " + str(STEP_DISTANCE) + ", BATCH_SIZE: " +
str(BATCH_SIZE) + ", EPOCHS: " + str(EPOCHS)


def build_classifier(input_shape, num_classes):
    model_m = Sequential()
    model_m.add(Reshape((TIME_PERIODS, 3), input_shape=(input_shape,)))
    model_m.add(Dense(100, activation='relu'))
    model_m.add(Dense(100, activation='relu'))
    model_m.add(Dense(100, activation='relu'))
    model_m.add(Flatten())
    model_m.add(Dense(num_classes, activation='softmax'))
    return model_m


def train_classifier(input, output, input_shape, num_classes):
    model = build_classifier(input_shape, num_classes)

    callbacks_list = [
        keras.callbacks.EarlyStopping(monitor='accuracy', patience=1)
    ]

    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy

    # Enable validation to use ModelCheckpoint and EarlyStopping callbacks.
    history = model.fit(input,
                        output,
                        batch_size=BATCH_SIZE,
                        epochs=EPOCHS,
                        callbacks=callbacks_list,
                        validation_split=0.2,
                        verbose=1)

    return history, model


def show_confusion_matrix(validations, predictions, model_m):
    matrix = metrics.confusion_matrix(validations, predictions)
    plt.figure(figsize=(6, 4))
    sns.heatmap(matrix,
                cmap='coolwarm',
```

```
                linecolor='white',
                linewidths=1,
                xticklabels=LABELS,
                yticklabels=LABELS,
                annot=True,
                fmt='d')
    plt.title('Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()

    precision_per_class, recall_per_class, f_score_per_class, _ = metrics.precision_rec
    (validations, predictions, zero_division=0)
    _, recall_avg, f_score_avg, _ =
    metrics.precision_recall_fscore_support(validations, predictions,
    pos_label=None, average="weighted", zero_division=0)
    mcc = metrics.matthews_corrcoef(validations, predictions)
    balanced_accuracy = metrics.balanced_accuracy_score(validations, predictions)
    accuracy = metrics.accuracy_score(validations, predictions)

    print('\nAccuracy on test data: %0.2f' % accuracy)

    return matrix, precision_per_class, recall_per_class, f_score_per_class, recall_avg


def main():
    if db_enabled:
        database = db.connect(db.DB_NAME, db.USER, db.PASSWORD, db.HOST, db.PORT)
        experiment_class, activity_class = db.get_classes(database)

    df = create_dataframe(FILES)

    max_test_predictions = []
    max_y_tests = []
    f1s = []
    mccs = []
    accs = []

    uid = uuid.uuid4()
    for s in SUBJECTS:
        print("Test subject: "+str(s))
        print("Loading test and train set...")
        xtrain, ytrain, xtest, ytest, input_shape, num_classes = preprocess(df, s)
        print("Test and train are ready to go")

        print("Training classifier...")
```

```
        history, model = train_classifier(xtrain, ytrain, input_shape, num_classes)
        test_prediction = model.predict(xtest)

        max_test_prediction = np.argmax(test_prediction, axis=1)
        max_y_test = np.argmax(ytest, axis=1)
        max_test_predictions.extend(max_test_prediction)
        max_y_tests.extend(max_y_test)

# print confusion matrix and get evaluation results of the test data
        matrix, precision_per_class, recall_per_class, f_score_per_class,
        recall_avg, f_score_avg, mcc_score, balanced_acc, acc
        = show_confusion_matrix(max_y_test, max_test_prediction, model)
        f1s.append(f_score_avg)
        mccs.append(mcc_score)
        accs.append(acc)

    show_confusion_matrix(max_y_tests, max_test_predictions, model)
    print("Statistics overview:")
    print("F1-scores:")
    print(f1s)
    print("Mcc scores:")
    print(mccs)
    print("Accuracies:")
    print(accs)


if __name__ == '__main__':
    main()
```

10.2 Traing The Variational Autoencoder

For this, the same VAE class is used as in the pipeline.

```
if __name__ == "__main__":
    m, loss = model_train(data)
    torch.save(m, "VAEmodel.bin")
```

10.3 Plotting Latent Representations Of Variational Autoencoder

```
import pickle
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D\

m = torch.load("VAEmodel.bin")

res = []
```

```
for i, window in enumerate(data):
    res.append(m(torch.unsqueeze(data[i], 0)))
zipped = list(zip(res, ytrain))

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

xs = [zipped[i][0][1][0][0].detach() for i, filled in enumerate(zipped)]
ys = [zipped[i][0][1][0][1].detach() for i, filled in enumerate(zipped)]
zs = [zipped[i][0][1][0][2].detach() for i, filled in enumerate(zipped)]

color_code = ytrain.tolist()
colors = {'0':'red', '1':'green', '2':'pink', '3':'yellow', '4':'orange', '5':'pink'}

scatter = ax.scatter(xs,ys,zs, c=color_code)
ax.view_init(200, 0)
pickle.dump(fig, open('alldata.pickle','wb'))
plt.show()
```