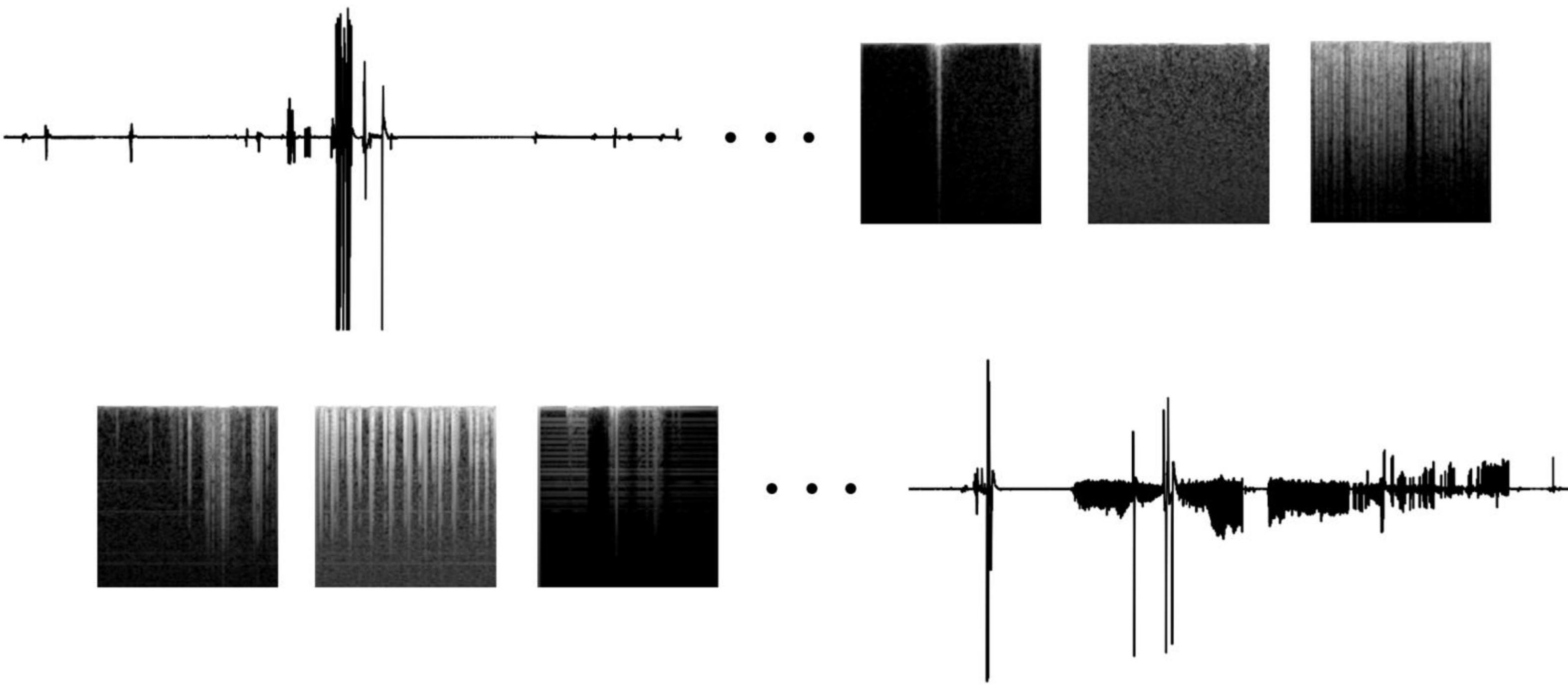
Unsupervised Learning for Automatic Classification of Needle Electromyography Signals

Sterre de Jonge

2023



Unsupervised Learning for Automatic Classification of Needle Electromyography Signals

Sterre de Jonge
Student number: 4477464
January, 2023

Thesis in partial fulfilment of the requirements for the joint degree of Master of Science in

Technical Medicine

Leiden University - Delft University of Technology - Erasmus University Rotterdam

Master thesis project (TM30004, 35 ECTS) for the track Sensing and Stimulation
Clinical Neurophysiology, Department of Neurology, Amsterdam UMC, location AMC &
Faculty of Mechanical, Maritime and Materials Engineering (3mE), TU Delft
January 2022 - January 2023

Supervisors:
Dr. C (Camiel) Verhamme
Dr. W.V. (Wouter) Potters
Dr.ir. W. (Winfred) Mugge

Thesis Committee members:
Prof.dr. M.R. (Martijn) Tannemaat (chair)
Dr. C. (Camiel) Verhamme
Dr.ir. W. (Winfred) Mugge
Dr. W.V. (Wouter) Potters

Preface and Acknowledgements

I'm very happy to be able to present the research that I have been working on over the past year. This research marks the end of my master's studies and, with it, my time as a student in Delft. While it's a bit sad to say goodbye to this phase of my life, I am excited to see what the future has in store. I have truly enjoyed working on this research with Wouter and Camiel over the past year and I am grateful for the lengthy discussions we had about the topic, which kept me motivated and focused. Thank you for your trust in me and the freedom you granted me to conduct this research. I have become more confident in myself. I also want to thank Winfred for his contribution, even though we didn't frequently meet the past year, the sessions we did have were very useful and I appreciate the feedback you gave me. Finally, a big thank you to my friends and family who have supported me and kept me sane throughout this whole process. Your encouragement means a lot to me!

Sterre de Jonge

Delft, 6th of January, 2022

Abstract

Introduction. Needle electromyography (EMG) is a diagnostic tool used to identify and localise neuromuscular disorders, but the current evaluation by a neurologist is subjective and requires years of training. Artificial intelligence can be used to automatically classify needle EMG signals. However, previous studies in this area have been subject to bias and have overestimated their performance. As a result, we aim to develop a clinically applicable method for automatically evaluating needle EMG signals to provide a more objective and efficient means of analysis. **Methods.** In this study, we implemented and evaluated two unsupervised learning models, a convolutional autoencoder + k-means clustering model and a deep convolutional embedded clustering model. The models were evaluated on two classification tasks: the classification of spontaneous, voluntary and insertional activity and the classification of spontaneous activity in needle EMG data classified as rest. We used hospital-acquired needle EMG data from the Amsterdam University Medical Centre (UMC), location AMC, from a total of 326 patients. The data was converted to Mel spectrograms, resulting in a total of 1.3 million images available for training. **Results.** The unsupervised learning models reached 93.5% accuracy on unseen test data. The classification of phenomena present in rest, such as fibrillation potentials and positive sharp waves, was less successful and requires further research. **Conclusion.** Our study provides valuable insights for the use of unsupervised learning in the automatic classification of needle EMG signals and highlights the need for further research in this area.

1 Introduction

Needle electromyography (EMG) is crucial for the identification and localisation of neuromuscular disorders [1–3]. A needle is placed in the muscle to record the activity of muscle fibers in resting and contracting states, with high spatial resolution. Detection of abnormalities provides diagnostic information upon a defect in the neuromuscular control pathway and thus, neuromuscular disorders.

In current practice, evaluation of needle EMG recordings is based on audio-visual interpretation of insertional, spontaneous and voluntary needle EMG signals by a neurologist [1]. Spontaneous activity in resting state, caused by individual muscle fibers or a motor unit, are described in guidelines after similar audio events. For example, "rain on a tin roof" for fibrillation potentials and "seashell" sound for endplate noise [2]. The presence of spontaneous activity is considered normal directly after insertional activity and when caused by activity near the endplate zone. Abnormal spontaneous activity may indicate pathologies at different levels of the motor unit [2]. During voluntary contraction, muscle fibers within a motor unit are activated which is recorded as a motor unit action potential (MUAP). The morphology and firing pattern of MUAPs help differentiate between two main categories of neuromuscular disorders, myopathic and neurogenic disorders [2].

Correct interpretation and recognition of pathologies in the needle EMG signal requires years of training before it is mastered by the neurologist. The interpretation can therefore be subjective, which is reflected in low interrater agreements. Depending on type of (pathological) waveform or level of training the interrater agreements are found to be between 47 and 91% [4, 5]. Techniques for objective analysis that allow a standardised yet sensitive and specific evaluation of needle EMG signals is therefore desired.

Recent studies focused on the automatic classification of needle EMG signals have taken advantage of the advances in the field of artificial intelligence (AI). All of the following studies reported a minimal classification accuracy of 95% for the classification of needle EMG signals. Torres-Castillo et al. used linear discriminant analysis to select two most discriminant time and time-frequency features for the classification of healthy, myopathy and neuropathy signals using *k*-nearest neighbour classifier [6]. Bose et al. retrieved features using weighted visibility graph and classified healthy, myopathy and neuropathy signals using naive-Bayes classifier [7]. Samanta et al. used a pre-trained residual network with 50 layers (ResNet50) to select features and classified healthy vs. diseased (myopathy or neuropathy) signals using support vector machine and *k*-nearest neighbour classifier [8]. Kamali et al. applied multiple instance learning to discern healthy, myopathic and neuropathy muscles [9].

Other studies transformed needle EMG signals to a two-dimensional spectrogram image to benefit from the advances made in deep learning for images. Sengur et al. used a convolutional neural network (CNN) to classify healthy and neuropathy spectrograms [10]. Nodera et al. converted the needle EMG signal to a Mel spectrogram [11]. The values in a Mel spectrogram are placed on the Mel scale which reflects human perception of audio signals [12–14]. The use of the Mel scale in the study by Nodera et al. aimed to mimic the evaluation process of a neurologist, who identifies abnormalities by sound (as well as by visual pattern recognition). A pre-trained Resnet50 model was used for the classification of spontaneous activity in resting state needle EMG signals and training was performed on artificially augmented Mel spectrograms in order to improve the classification accuracy [11]. Nodera et al. were the first to apply AI to rest needle EMG signals.

The high classification accuracies that were reported in above-mentioned studies could be a result of bias and overfitting, where the performance is overly estimated on the test set. External validation, which is important to recognise potential biases, was in none of the studies reported. A bias that was identified in several studies was the use of test data to inform decisions at several phases of model development, including feature selection, hyperparameter tuning and model selection. Additionally, these studies used small datasets containing signals collected in a controlled environment from a small number of patients. Online EMGLab database (`http://www.emglab.net`) was in multiple studies used [6–8, 10], this database consists of clean EMG signals without artefacts and a clear distinction between normal and pathological EMG signals [15]. Current literature on the classification of needle EMG signals has therefore several drawbacks that make clinical implementation difficult: (1) reliance on manually selected segments of data as a result of labelling, (2) lack of validation on independent data, and (3) use of a small subset of patient groups.

This study aims to develop a clinically applicable method to automatically evaluate needle EMG signals. This is achieved through (1) directly training models with hospital acquired needle EMG data (containing insertional, spontaneous and voluntary signals), and (2) perform training with vast amounts of data through the implementation of unsupervised learning techniques. This approach has not yet been implemented in previous literature. We will focus on the classification of spontaneous needle EMG signals, because earlier work primarily focused on the classification of voluntary signals. In order to do so, it is necessary to identify resting state signals from insertional activity and voluntary contraction in hospital acquired needle EMG signals. The unsupervised learning methods that we will implement are discussed in the next section, followed by Section 3 which describes the research question of this study.

2 Background

2.1 Clinical Needle EMG Examination

Needle EMG examination is performed at the hospital to identify neuromuscular disorders. A needle is inserted into the muscle to measure the electrical activity of that muscle. There are three stages of the examination: (1) the muscle's response immediately after needle insertion, (2) the muscle's activity when it is at rest (relaxed), and (3) the muscle's activity during voluntary contraction, from low to near maximal contraction [2]. It is normal for the muscle to have some response to the needle, but if the activity lasts longer than a few hundred milliseconds (known as increased insertional activity), it may indicate a problem [2]. It is, furthermore, expected that the muscle is electrically silent when it is at rest. Spontaneous activity that is observed for longer than 2-3 seconds is abnormal [2]. The exception is when the needle is near the endplate zone (resulting in endplate noise/spikes), this activity is normal. The following spontaneous waveforms may be observed during muscle rest: fibrillation potentials, positive sharp waves (PSW), complex repetitive discharges (CRD), myotonic discharges, fasciculation potentials, myokymic discharges, and endplate noise/spikes. Fibrillation potentials and PSW have the same significance and they are both spontaneous depolarisation of a muscle fiber [2]. The examination is completed by evaluating MUAPs during voluntary contraction. The muscles examined during a routine clinical examination are determined by the clinical question and the patient's symptoms.

2.2 Unsupervised Learning Techniques

In unsupervised learning, a model is trained without providing any labelled outcomes or target variables. Classification is performed by finding intrinsic differences in the data. Clustering is a method to achieve this, but clustering methods perform poorly in high-dimensional spaces [16]. Dimensionality reduction is therefore used in unsupervised learning to improve clustering.

A recently introduced method to achieve dimensionality reduction and feature extraction, simultaneously, is through autoencoders [17, 18]. Autoencoders are types of deep learning models that are trained to learn efficient representations of data. They are designed to compress input data into a smaller and more compact form while still retaining as much of the important information as possible.

An autoencoder consists of two parts: an encoder and a decoder. The encoder, described by $z = f(x)$, maps the input data (x) to a lower-dimensional representation (z), called the latent space or representation. The decoder, described by $r = g(z)$, uses the latent representation to reconstruct the original input data (reconstruction r) [19]. The model is trained by minimising the reconstruction error, which is the difference between the input and the output. A meaningful latent space z is achieved because the network is forced to capture the most salient features of the training data in a lower-dimensional space.

Autoencoders may consist of different kinds of building blocks to handle different kinds of input data. Convolutional autoencoders consist of convolutional layers and are capable to handle image data as input data. Convolutional layers preserve the spatial structure of images and are very efficient in identifying local and global structures. These layers are widely and successfully used in convolutional neural networks in the medical field, such as the classification or segmentation of medical images [20–22]. Transformation of needle EMG data to images (such as Mel spectrograms) allows us to implement convolutional autoencoders.

A convolutional autoencoder (CAE) is presented in Guo et al. [18] to learn ten features from unlabelled images of MNIST database (28×28 images of handwritten digits). These features were subsequently used in *k*-means clustering, a popular clustering method [23], to form ten clusters representing ten digits (0-9).

The goal of *k*-means clustering is to minimise within cluster-variances (inertia). The algorithm consists of the following steps: (1) initial centroids (number = *k)* are randomly chosen, (2) all samples are assigned to its nearest centroid, (3) centroids are updated by taking the mean value of all the samples that are assigned to the prior centroid. Step two and three are consecutively repeated until the centroids do not move significantly. The outcome is dependent on the (random) initialisation and it is thus common practice to initialise multiple times to reach a global optimum.

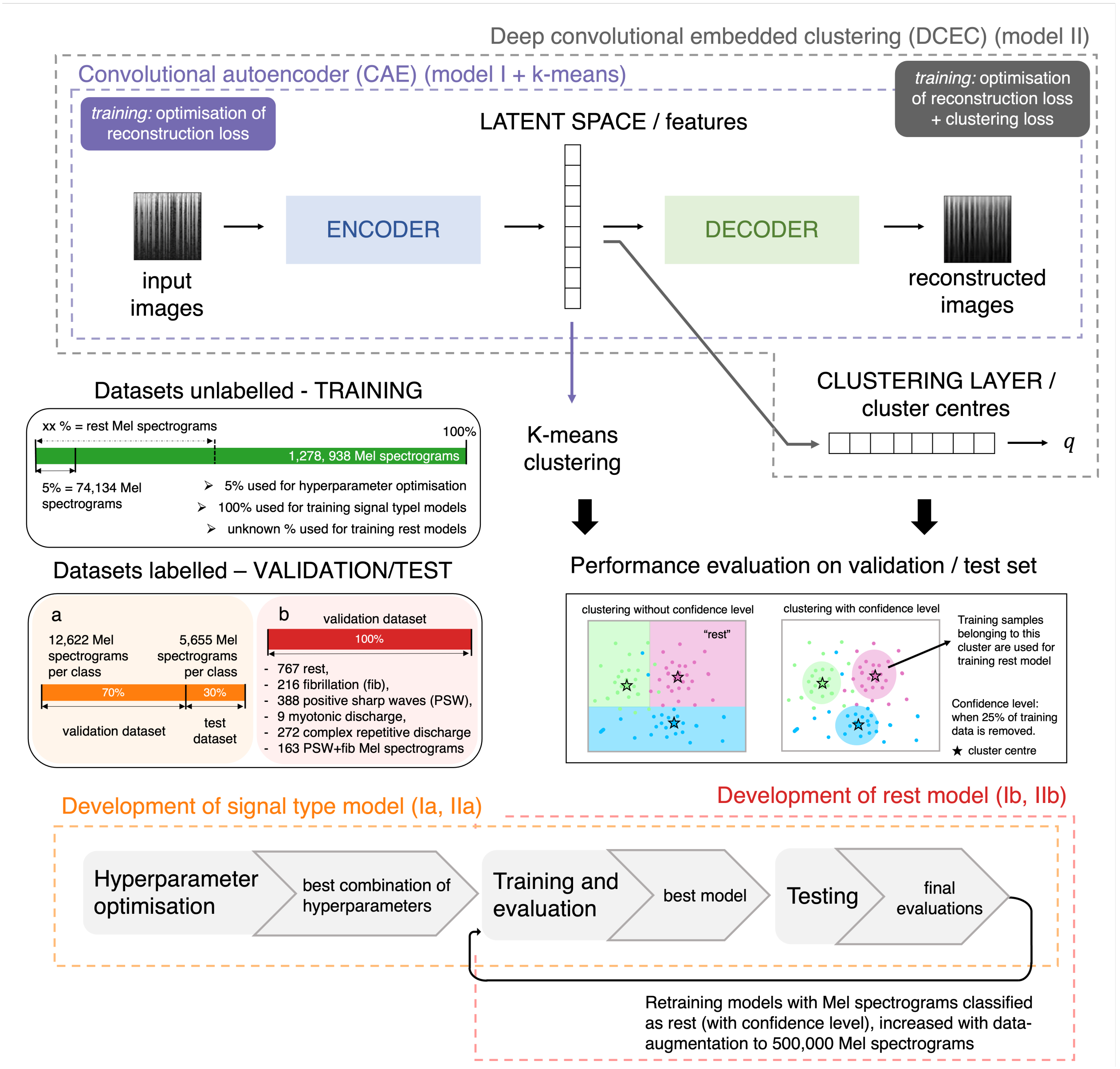
In the CAE + *k*-means clustering approach, features are learned to reconstruct the input by minimising the reconstruction loss. This may not lead to the most optimal feature representation that is used during clustering, aiming to identify the classes. It could be that different features are learned from images that are visually alike, if both sets of features still lead to a correct reconstruction. The authors therefore added the clustering loss, resulting in the approach named deep convolutional embedded clustering (DCEC) [17, 18]. The clustering loss dictates how well the data is mapped to *k* cluster centres based on an auxiliary target distribution. The target distribution puts more emphasis on data points that were assigned with high confidence. The purpose of the clustering loss is to improve cluster purity and hence strengthen predictions [17].

3 Research Question

The aim of this study is to implement two unsupervised learning strategies, convolutional autoencoder (CAE) followed by *k*-means clustering (strategy I) and deep convolutional embedded clustering (DCEC) (strategy II) and evaluate their performance on the classification of needle EMG signals converted to Mel spectrograms.

The following research question is addressed: *What is the accuracy of unsupervised learning strategies in the automatic classification of needle EMG signals?*

We hypothesise, that: (i) unsupervised learning techniques can be used to classify rest, contraction and needle Mel spectrograms with a minimal classification accuracy of 95% (classification task a), (ii) unsupervised learning techniques are less effective at classifying rest phenomena (i.e.: fibrillation potentials, positive sharp waves, complex repetitive discharge and myotonic discharge) in Mel spectrograms classified as rest, with a minimal classification accuracy of 70% (classification task b), (iii) DCEC performs better compared to CAE + k-means clustering in both classification tasks.



Deep convolutional embedded clustering (DCEC) (model II)
Convolutional autoencoder (CAE) (model I + k-means)
training: optimisation of reconstruction loss
training: optimisation of reconstruction loss + clustering loss
LATENT SPACE / features
input images
ENCODER
DECODER
reconstructed images
CLUSTERING LAYER / cluster centres
q
K-means clustering
Datasets unlabelled - TRAINING
xx % = rest Mel spectrograms
100%
1,278, 938 Mel spectrograms
5% = 74,134 Mel spectrograms
5% used for hyperparameter optimisation
100% used for training signal typel models
unknown % used for training rest models
Datasets labelled – VALIDATION/TEST
a
12,622 Mel spectrograms per class
5,655 Mel spectrograms per class
70%
30%
validation dataset
test dataset
b
validation dataset
100%
- 767 rest,
- 216 fibrillation (fib),
- 388 positive sharp waves (PSW),
- 9 myotonic discharge,
- 272 complex repetitive discharge
- 163 PSW+fib Mel spectrograms
Performance evaluation on validation / test set
clustering without confidence level
"rest"
clustering with confidence level
Training samples belonging to this cluster are used for training rest model
Confidence level: when 25% of training data is removed.
★ cluster centre
Development of signal type model (Ia, IIa)
Development of rest model (Ib, IIb)
Hyperparameter optimisation
best combination of hyperparameters
Training and evaluation
best model
Testing
final evaluations
Retraining models with Mel spectrograms classified as rest (with confidence level), increased with data-augmentation to 500,000 Mel spectrograms

Figure 1: Structure of the proposed method. Convolutional autoencoder (in purple) is, combined with k-means clustering, the first (I) model. Deep convolutional embedded clustering (in grey) is the second (II) model. The output of model I is a trained k-means clustering method and the output of model II is a set of soft probabilities indicating the likelihood of a data sample belonging to each cluster (q). Output of both models is evaluated using validation/test set. True label of cluster is determined by majority of samples belonging to a class in the validation/test set. Clustering with confidence is applied to select data samples that meet a certain confidence level, this level is determined when 25% of the data is removed from training dataset. The development of models for each classification task is shown in the flow diagram at the bottom. The corresponding labelled datasets (a, b) are shown in middle left box. All training data (green) is used for development of signal type model, Mel spectrograms that are classified as rest with confidence level are used for development of the rest model.

4 Methods

In this study, two models were implemented [18]: convolutional autoencoder (CAE) + *k*-means clustering (I) and deep convolutional embedded clustering (DCEC) (II). These models were then applied to two subsets of clinical data: all data (a) and rest data only (b). First, models I and II were trained to classify all needle EMG signals in the muscle state (rest or contraction) or presence of needle movement; leading to signal type models. Second, the samples that were classified as rest were used to train the models to identify phenomena that were present in rest, leading to rest models. The methods section is summarised in Figure 1, each aspect will be discussed in detail in the following sections.

4.1 Data Retrieval and Preparation

4.1.1 Study Population

Clinical needle EMG recordings are digitally stored from patients undergoing routine clinical examinations at the neurophysiology department of Amsterdam UMC, location AMC. Data recordings from all muscles are available that were examined during clinical care. For this research, we retrospectively exported recordings from 326 patients between December 2016 and April 2020. This data was anomalously handled according to the General Data Protection Regulation (GDPR). The data consists of needle EMG recordings, muscle name, and pathological phenomena for each muscle as manually annotated during the needle EMG exam (e.g., the presence and intensity of fibrillation potentials, the presence and intensity of positive sharp waves, and normal or increased insertional activity). Selection did not take place during exportation of recordings. All (raw) recordings were exported as uncompressed waveform audio files (wav) with a sample rate of 44.1 kHz.

4.1.2 Data Acquisition

Needle EMG examination was performed using a needle and Synergy software. Signals were sampled with a sample frequency of 44.1 kHz and the signals were band-pass filtered (10 Hz - 10 kHz) during acquisition. The patient was asked to relax and contract the muscle, to record signals in both states. Signal quality was optimised by re-positioning the needle during the exam. Multiple recordings were made per patient and/or muscle, if this was clinically required, and each recording had a maximal duration of 120 s.

4.1.3 Preprocessing

The raw recordings (wav files) were converted to Mel spectrograms during preprocessing. The Mel scale places frequencies on a logarithmic scale where equal distances are perceived as equal by the human ear [24]. A similar approach reported by Nodera et al. [11] was adopted for the computation of the Mel spectrograms.

The root mean square of the data was normalised to -26 dB using the Python package Pyloudnorm (version 0.1.0) [25]. The files were then sampled to 1.48 s overlapping segments using a sliding window and step size of 0.1 s. Extraction of segments with 1.48 s duration resulted in a square-shaped Mel spectrogram. Step size is arbitrary but a relatively small step size of 0.1 s ensures that The Mel spectrogram was then computed using the Librosa toolbox in Python (version 0.8.1) [26] with the following parameter settings: y = data samples, sample frequency = 44 100, fmax = 10 000, number of Mel bands = 128, hop length = 512. In words, the computation of the Mel spectrogram consisted of the following steps: (1) the 1.48 s data sample was sampled to 128 non-overlapping 11.6 ms windows (containing 512 data points per window), (2) the short-time Fourier transform was used to calculate a magnitude spectrum for each window, (3) the Fourier transformed signal was then filtered through a set of band-pass filters, known as the Mel filter bank. An example of a Mel filter bank is shown in Figure 2 depicting six (triangular) filters (compared to 128 filters that were implemented). The filters are more discriminative at lower frequencies than at higher frequencies, thus placing the frequencies on a Mel scale. The filters are triangular shaped and have a response of 1 at the centre frequency and this decreases linearly towards 0 until it reaches the centre frequency of the adjacent filter.

There are some limitations to this approach because the pixel values (representing the values of the Mel spectrogram) are scaled according to the values in the 1.48 s time signal. The effect of this is shown in Appendix D.4. It was for this thesis not feasible to change this approach because we "discovered" it late in the process.

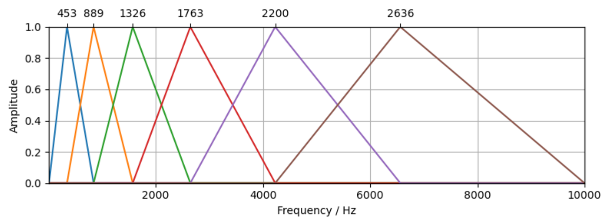


Figure 2: Example of Mel filter bank. Each coloured triangle corresponds to a band-pass filter that is applied to place values from spectrogram onto the Mel scale.

The final steps applied to the Mel spectrogram included converting the values to decibels, normalising the values using min-max normalising, and scaling the values to the range [0,255] so that the Mel spectrogram could be saved as an image. The resulting Mel spectrogram (with size 128 $\times$ 128) was used as the image input for all the models discussed in the following sections. When using deep learning, it is important to scale the images to the range [0,1]. Therefore, the images were rescaled to this range when they were loaded during training.

4.1.4 Datasets and Labelling

The datasets that were used during each stage of classifying needle EMG signals are shown in Figure 3. Unlabelled data was used for training and labelled data for validation and testing. Remaining (and unlabelled) files from patients in the labelled dataset were excluded from the unlabelled dataset to truly separate training and validation/test set. Data is best disjoint at patient-level and not at signal-level to reduce overfitting and bias [27]. Data recorded from bulbar muscles were excluded during training, because waveforms in these muscles possess different characterisations compared to other skeletal muscles [2].

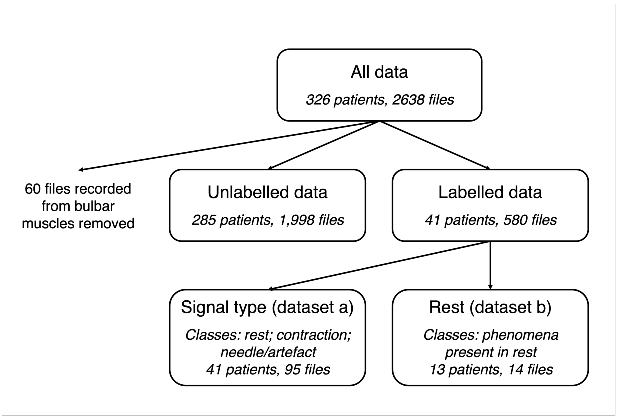


Figure 3: Division of labelled and unlabelled data from available database. Patients whose files were used in the labelled datasets were excluded from the unlabelled dataset . Files recorded from bulbar muscles were removed.

Dataset unlabelled
Unlabelled files were used for training. Mel spectrograms were computed from a total of 1,998 wav files from 285 patients.

Dataset a: Signal Type
95 files from 41 patients were annotated for classes rest, contraction and needle. Annotation was performed using an in-house developed annotation tool (in Python 3.9) that allowed for both auditory and visual inspection of the needle EMG signal. The files were annotated by at least two examiners of four examiners in total. Appendix A.1 describes the rules that were followed during annotation. Nine files of 120 s were labelled by all (four) of the examiners to determine the interrater reliability [1] The final annotation (reference standard) was determined when both examiners reached the same class label. The class labels were assigned to 1.48 s segments when the following requirements were met:

- 100% of a segment labelled as rest receives rest label
- 100% of a segment labelled as contraction receives contraction label
- ≥15% of a segment labelled as needle movement receives needle label

For the needle movement class label, the requirement was determined to be met when 15% of the annotation received the needle label, as needle movement is generally a short-duration activity. The other two classes were created using full-length annotations (100%) to avoid contamination with other classes in the resulting Mel spectrograms.

Dataset b: Rest
14 files from 9 patients were annotated for class rest and spontaneous activity in rest: fibrillation potentials, PSW, CRD, and myotonic discharge. These files were selected for this dataset when a note was made of the presence of these phenomena during examination, as some of them are rare and the purpose of this dataset is to provide Mel spectrograms of these classes. The in-house developed annotation tool was updated to allow annotation for multiple additional classes (elaboration in Appendix A.2). The files were annotated by one experienced examiner. The rules followed during annotation are described in appendix A.3. The class labels were assigned to 1.48 second segments when the following requirements were met:

- 100% of a segment labelled as rest receives rest label
- ≥5% of a segment labelled as spontaneous activity (fibrillation potentials, PSW, CRD, or myotonic discharge) and remaining portion is labelled as rest receives spontaneous activity (fibrillation potentials, PSW, CRD, or myotonic discharge) label
- ≥5% of a segment is labelled as PSW and ≥5% of the segment is labelled as fibrillation potentials and remaining portion is labelled as rest receives PSW + fibrillation potentials label

The rationale for these requirements is that spontaneous activity can be of short duration. Annotation was performed from starting to end point of waveform and the spontaneous activity class labels were therefore given to segments containing 5% of specific phenomena alongside normal rest signal. As fibrillation potentials and PSW have the same clinical significance, the final class was created when they appeared in the same segment.

4.2 Model I: Convolutional Autoencoder + *k*-means Clustering

The convolutional autoencoder (CAE) was implemented in Python using Keras (version 2.8.0) [28]. The parameters of the CAE were updated with mini-batch gradient descent using the Adam optimisation algorithm. The loss function optimised was the mean squared error (MSE), which is calculated over the input (y_i) and output ($\tilde{y_i}$) as follows:

$$L_r = MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y_i})^2$$

An example of the network is shown in Figure 4. The network shown here consists of three convolutional layers and latent space z that has size 64. This CAE shown here is therefore trained to learn 64 features.

[1]This was performed in a previous study by Deborah Hubers (master thesis title: Artificial Intelligence-based Classification of Electromyography, date: April 2022) (link to paper not available).

The features in the latent space were subsequently used to train the k-means clustering method from Scikit-Learn package (version 1.0.2) [29]. Number of clusters were determined by hyperparameter optimisation (classification task a) and by the maximum number of classes (classification task b).

4.3 Model II: Deep Convolutional Embedded Clustering

The deep convolutional embedded clustering (DCEC) was implemented in Python with Keras, as well. The DCEC model was end-to-end trained with simultaneous optimisation of the reconstruction loss (L_r) and clustering loss (L_c). The network is shown in Figure 5 and is composed of CAE and a clustering layer which is connected to the latent space of CAE.

The clustering layer iteratively refines the clusters by learning from their high confidence assignments. This is realised by comparing the soft assignment of all training data with an auxiliary target distribution. The soft assignment is computed using the Student's t-distribution. For every input image (a sample i) the similarity is measured between embedded point z_i (representing features in the latent space of a input image x_i) and centroid μ_j [17]:

$$q_{ij} = \frac{(1 + \|z_i - \mu_j\|^2)^{-1}}{\sum_j (1 + \|z_i - \mu_j\|^2)^{-1}}$$

Where q_{ij} is the probability of assigning sample i to cluster j. The target distribution for each sample p_i is computed by raising q_i to the second power and then normalising by frequency per cluster [17]. For sample i and cluster j the target distribution is:

$$p_{ij} = \frac{q_{ij}^2 / \sum_i q_{ij}}{\sum_j (q_{ij}^2 / \sum_i q_{ij})}$$

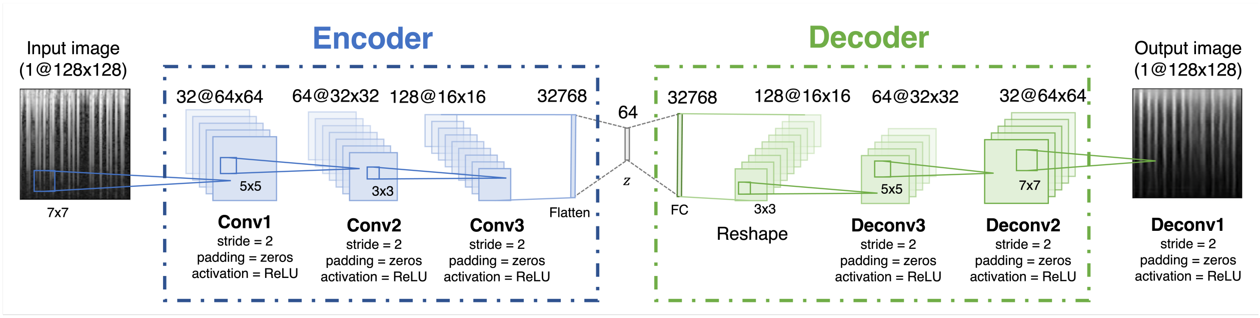


Figure 4: Convolutional autoencoder adopted from Guo et al. [18]. It takes a black and white image as input (in this case, a Mel spectrogram) and learns a representation of a set of pixels at each layer using a filter (the size is the depth of the layer multiplied by the kernel size). The first convolutional layer learns 32 representations and, since the input image has a depth of 1, the filters have a size of $1 \times 7 \times 7$. The second layer increases the number of representations to 64 and the filters have a size of $32 \times 5 \times 5$ (because the depth has increased to 32 layers). The number of representations increases with the depth of the autoencoder because more complex representations can be learned using the input from the previous layer. Zero padding and a stride of 2 are applied, which decreases the feature maps (each representation) by half at each layer. After the final convolutional layer, all values are flattened into a one-dimensional layer with $32,768 values$, which is then reduced to the latent representation / features (z). The decoder is symmetrical to the encoder and learns to reconstruct the input using these features. The learning process occurs by minimising a reconstruction loss between the input and output image.

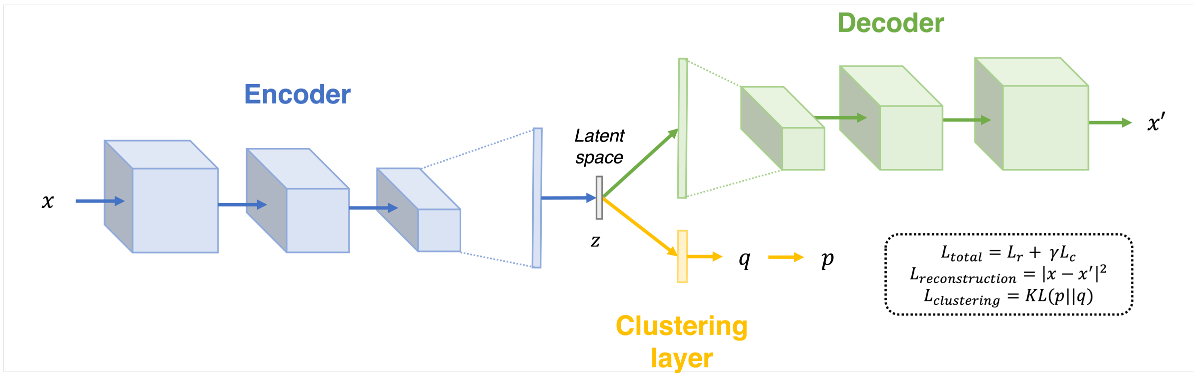

Encoder
Decoder
x
Latent space
z
x'
q
p
Clustering layer
$L_{total} = L_r + \gamma L_c$
$L_{reconstruction} = |x - x'|^2$
$L_{clustering} = KL(p||q)$

Figure 5: Deep convolutional embedded clustering adopted from from Guo et al. [18]. It is composed of the convolution autoencoder (encoder + decoder) that takes an input image as input (x) and reconstructs this using the learned features to the output/reconstructed image (x'). The latent space is additionally connected to the clustering layer representing the cluster centres. The output of the clustering layer is q, representing the soft probabilities of a data sample belonging to each cluster. q computed over all training samples is used to compute auxiliary target distribution p. The model is end-to-end trained by optimising the reconstruction loss and clustering loss simultaneously.

The target distribution serves as the ground truth soft label, but since it depends on predicted soft label, it needs to be updated during training. This should not occur at each iteration (to avoid instability [18]) and the target distribution is therefore updated at the beginning of an epoch. The clustering loss is defined as the Kullback-Leibler divergence loss between the soft assignments q_i and the auxiliary distribution p_i as follows:

$$L_c = KL(P\|Q) = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}}$$

DCEC is trained by first pretraining the CAE (without the clustering loss) to obtain meaningful target distribution. The cluster centres are after pretraining initialised by performing *k*-means on embedded features of all training data (with 100 random initialisations). The clustering layer maintains cluster centres as trainable weights. The model is then updated by both the clustering loss L_c and reconstruction loss L_r as follows:

$$L_{total} = L_r + \gamma L_c$$

The clustering loss is multiplied by a value γ that is smaller than one because predominance of clustering loss in the total loss may lead to corruption of the feature space. In previous work it was shown that preservation of local structures helps to stabilise the training procedure [18].

Model I and II were both implemented using publicly available code from Guo et al. [18]. Changes were made to the original code to adapt it to our user case. The code can be found in the GitHub (`https://github.com/Sterre26/Unsupervised_Learning_Needle_EMG_Classification`). Some parts of the original code, with regards to the implementation of the clustering layer, had to be revised because the original implementation was not suitable (i.e. less efficient, memory problems) for the amounts of data that we are applying it to. Elaboration on these changes is provided in Appendix B.

4.4 Clustering with Confidence

The output of both models is a cluster assignment where all data samples are clustered in *k*-dimensional space (with *k* the size of latent space/ number of features selected in hyperparameter optimisation). An additional step was introduced where data samples are clustered with a confidence level, which quantifies the level of certainty that a data point belongs to its assigned cluster. Data points that do not strongly belong to their assigned cluster were removed. The likelihood that a data sample (x_i) belongs to cluster j was calculated with the following formula [30]:

$$w_{ij} = \frac{1}{\sum_{k=1}^{c}(\frac{||x_i - c_j||}{||x_i - c_k||})^2}$$

With w_{ij} the weights and c_j the coordinates of the centre of the jth cluster. The sum of weights is 1 for each data sample. When the number of clusters is higher than the number of class labels, the weights for a data sample are combined from clusters belonging to the same class. The confidence level is determined by removing 25% of the data samples from the training set.

4.5 Development of Signal Type Model (Ia, IIa)

The first stage of development consisted of training models for the classification of needle EMG signals in rest, contraction and needle classes to identify spontaneous activity from insertional activity and voluntary contraction. The training process consisted of three phases, shown in Figure 1 within the orange coloured box. All three phases were completed for model I and model II, separately. The following paragraphs discusses each phase in more detail. Training of the models during the stages described in the following paragraphs was performed on a Linux machine with 16-core CPU, 110 Gb RAM and Nvidia Tesla T4 GPU.

4.5.1 Hyperparameter Optimisation

The goal of hyperparameter optimisation is to select the best combination of hyperparameters that results in the highest performance (evaluated on the validation dataset). Hyperparameter optimisation took place for the hyperparameters shown in Table 1. The selection of those parameters was carefully made, detailed elaboration can be found in Appendix C. We tested the same set of hyperparameters on both models, despite the architectural similarities in the network. The rationale is that a different architecture or learning approach may be needed to update the weights from the clustering layer, that are included in the second model, as optimal as possible.

Hyperparameter optimisation was conducted with Hyperopt package (version 0.2.7) [31] using the Tree of Parzan algorithm, a Bayesian approach. In Bayesian model-based optimisation, a probability model of the objective function is built and iteratively used to select the most promising hyperparameters to evaluate in the true objective function. Fewer iterations are needed to reach the best combination of hyperparameters compared to, for instance, a standard grid search.

Each trial consisted of a newly selected set of parameters used to train a model for a maximal number of epochs. The number of epochs is not set as a hyperparameter because it is only desired to get a rough understanding of the performance. Models generally perform better when trained for a longer period of time (using more epochs) and with more data. This is, however, computationally not feasible during hyperparameter optimisation. Applying the same rationale to the amount of training data, hyperparameter optimisation was performed using 5% of the training data (see also Figure 1). Hyperparameter optimisation for model one was conducted for 1500 trials where 1500 models were trained for 8 epochs. Hyperparameter optimisation for model two was conducted for 800 trials and each trial consisted of pretraining CAE for 8 epochs followed by training DCEC for 8 epochs.

The performance was computed after each trial using validation data. This dataset contains 70% of Mel spectrograms of labelled dataset a (see also Figure 1), the dataset is balanced and surplus data was randomly removed. The objective function that was optimised by Hyperopt was the F1 score for samples classified as rest. F1 score is defined as the harmonic mean of precision and recall (sensitivity):

$$F1 = \frac{2}{\frac{1}{recall} + \frac{1}{precision}} = 2 * \frac{precision * recall}{precision + recall}$$

With precision and recall:

$$Precision = \frac{TruePositive(TP)}{TP + FalsePositive}$$

$$Recall = \frac{TP}{TP + FalseNegative}$$

The best combination of hyperparameters were selected for the model that resulted in highest F1 score for rest (on the validation set) after all iterations. The best combination of parameters that performed best for model I and II on classifying samples as rest from full-length hospital acquired needle EMG data, were also applied for the second task (classification of rest samples in phenomena present in rest). This was possible because the domain is very similar between the two datasets, so further hyperparameter optimisation is not necessary and the same set of parameters can be used for training. Hyperparameter optimisation, and training in general, requires a lot of resources (hyperparameter optimisation for both models took about two weeks each on a powerful computer), so it was not feasible to conduct this process a second time on a slightly different dataset.

Table 1: Hyperparameter search space with corresponding value(s) for hyperparameters that are tuned for model I (convolutional autoencoder + k-means clustering) and model II (deep convolution embedded clustering). Batch size and learning rate are coupled where different learning rates apply for the smaller or larger batch size. There are three different learning rate schedules. The number of layers in the convolutional autoencoder determine the size of the filters and kernels, as well. Selection of best combination of hyperparameters is shown on the right hand side.

Hyperparameters	**Value(s)**				**Selection model I**	**Selection model II**
Learning						
Batch size	64	256	64	256	64	64
Schedule	constant		cyclical without (clr1) or with (clr2) amplitude scaling		constant	PT: constant, T: clr1
Rate	1e-6, 5e-6, 1e-5, 5e-5, 1e-4, 5e-4 or 0.001	1e-4, 5e-4, 0.001, 0.005, or 0.01	[1e-7, 1e-4]	[1e-5, 0.001]	5e-5	PT: 0.001, T: [1e-7, 1e-4]
Optimiser	Adam				Adam	Adam
Architecture						
Layers	three	four	five		three	three
Filters	[16, 32, 64] or [32, 64, 128]	[16, 32, 64, 128] or [32, 64, 128, 256]	[8, 16, 32, 64, 128] or [16, 32, 64, 128, 256]		[16, 32, 64]	[32, 64, 128]
Kernels	[5x5, 5x5, 3x3] or [7x7, 5x5, 3x3]	[5x5, 5x5, 3x3, 3x3] or [7x7, 5x5, 5x5, 3x3]	[7x7, 5x5, 5x5, 3x3, 3x3] or [7x7, 7x7, 5x5, 3x3, 3x3]		[7x7, 5x5, 3x3]	[7x7, 5x5, 3x3]
Stride	2				2	2
Padding	”same” (with zeros)				”same”	”same”
Activation	ReLU, LeakyRelu alpha 0.1, LeakyRelu alpha 0.2, LeakyRelu alpha 0.3				ReLU	LeakyRelu alpha 0.2
Batch norm	True or False				True	True
Features	10, 12, 14, 16 18, 20, 24, 28, 32, 36, 40, 48, 56, 64, 128, 256, 512, 1024, 2048, or 4096				64	56
Clustering						
Clusters	5, 6, or 7				6	7
Gamma	0.05, 0.1, 0.2, 0.3, 0.4 or 0.5				N.A.	0.05

CLR = cyclical learning rate; MSE = Mean Squared Error; N.A. = not applicable; PT = pretraining; ReLU = Rectified Linear Unit; T = training.

4.5.2 Training and Evaluation

Model I and model II were both trained for a longer period of time with more data once the best combination of hyperparameters were selected. All training data was used during training. Both models were trained for 100 epochs, for model II this means that the model is firstly pretrained for 15 epochs and then trained for 85 epochs. The purpose of pretraining is to obtain meaningful target distribution which means that the CAE should be able to reconstruct meaningful images, but it is not necessary that the model fully converges. The following metrics were tracked during training and evaluated after each epoch:

1. Losses - Reconstruction loss for model I. Reconstruction, clustering and combined total loss for model II. Losses were evaluated both on training datasets and validation dataset.
2. Performance - Accuracy and F1 for rest was tracked during training on the validation dataset. True label of cluster was determined by majority of samples belonging to a class in the validation set. Multiple clusters can belong to one class.
3. Performance with confidence - The performance (accuracy and F1 for rest) was additionally computed only on data samples that were clustered with minimal confidence level. Confidence level was determined at each epoch that resulted in 25% data removal of training dataset. The performance with confidence, the confidence level and the percentage of removal on the validation dataset was tracked.
4. Delta label - This metric measures the similarity in clustering assignments between consecutive epochs and thus to what extend the same data sample results in the same prediction. It is determined by comparing previous clustering assignments with the current cluster assignments. The clusters (i.e. which number each cluster receives) remains the same for model II, because the cluster centres are fixed by the clustering layer. For model I it was necessary to reorder the cluster numbering, because this is randomly assigned by *k*-means clustering. We compared the current cluster centres with the previous cluster centres using pairwise minimal distance.

4.5.3 Testing

After training for 100 epochs, the final model was tested on a dataset that had not been seen during earlier stages of model development (the test dataset). This dataset contains 30% of Mel spectrograms of labelled dataset a (see also Figure 1), the dataset is balanced and surplus data was randomly removed. The model's performance was evaluated using precision, recall, F1-score, and accuracy. These performance metrics were depicted in a performance table and the cluster performance per cluster was additionally visualised in a confusion matrix, with true labels as the class labels and predicted labels as each cluster.

Clustering results were visualised on a T-distributed stochastic neighbour embedding (t-SNE) plot. This is a dimensionality reduction method that allows us to visualise clustering in a high dimensional space on a two-dimensional space. The features from test datasets were reduced to two using Scikit-Learn package (version 1.0.2) and with the following parameters for t-SNE: n_components=2, perplexity=150, init='pca', learning_rate='auto', n_iter=2000, n_iter_without_progress=150, n_jobs=2, random_state=0. The resultant two features were plotted in a two-dimensional figure.

4.6 Development Rest Model

The second stage of development consisted of training models to identify phenomena present in rest. The best signal type models (model I and model II) were used to identify rest Mel spectrograms, that were clustered with a confidence level, creating two rest training datasets. The labelled rest set was also clustered by both models, and Mel spectrograms that were not clustered as rest were removed from the validation set. The process of training the rest models is shown at the bottom in Figure 1. Hyperparameter optimisation was not performed for rest models, and best combination of parameters was re-used from development of signal type model.

4.6.1 Data-augmentation

Data-augmentation was applied to the validation dataset to increase and balance the classes. The rest validation dataset is small and to enable more freedom in evaluating the models with balanced classes we artificially increased all classes to a maximum of 2,000 Mel spectrograms per class. A similar approach reported by Nodera et al. [11] for data-augmentation was adopted. Data-augmentation was performed by skewing and distorting the Mel spectrograms. Data-augmentation was also applied to the rest training dataset, this dataset was artificially increased to 500,000 Mel spectrograms. This step was necessary to allow the model to learn on augmented images.

4.6.2 Training and Evaluation

Model I and II were both trained for 100 epochs using rest Mel spectrograms. Model II was again pretrained for 15 epochs and subsequently trained for an additional 85 epochs. Performance during training was evaluated for the following metrics: training and validation loss, clustering performance (accuracy and F1 for every class), and similarity in clustering assignments between consecutive epochs (delta label).

4.6.3 Testing

The final model was not tested on unseen test data, because this data was not available. However, the same steps were performed as performed on the signal type dataset (described in Section 4.5.3) but then on the validation dataset. The final model (after training for 100 epochs) was tested on the validation dataset with data-augmentation to compute precision, precision, recall F1-score, and accuracy. The testing results consists both of the performance table and the confusion matrix. Clustering results were not visualised using t-SNE.

4.7 Outcome Measures

Outcome measures were evaluated on best models after training for 100 epochs using labelled datasets. For development of signal type models (model I and II) this is a balanced dataset of 30% of labelled dataset a, the test dataset. For development of rest models (model I and II) this is the validation dataset for samples that were classified as rest and after applying data-augmentation to artificially increase to 2,000 images per class. Primary outcome measure is accuracy and secondary outcome measures are precision, recall and F1 score.

5 Results

5.1 Data

5.1.1 Dataset a: Signal Type

The annotation process resulted in an average exclusion of 24.8% of each data file. The excluded parts were the parts of the file where the two examiners disagreed or did not know the label. The interrater reliability of the different reviewers (evaluated on nine files) is 0.77[2]. Table F.1 in the appendix shows the interrater reliability of each file. Extraction of the 1.48 s segments with a sliding window of 0.1 resulted in 18,277 rest Mel spectrograms, 32,573 contraction Mel spectrograms and 26,405 needle Mel spectrograms. The Mel spectrograms in the classes were then split (on patient-level) in balanced validation and test dataset with respectively 12,662 Mel spectrograms per class and 5,655 Mel spectrograms per class. Examples of the resultant Mel spectrograms of each class are shown in Appendix D.1.

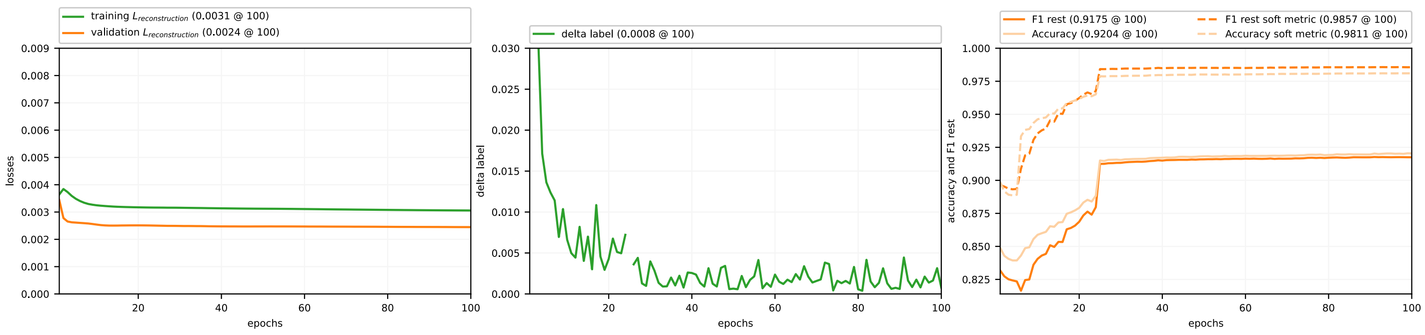


Figure 6: Training and evaluation results of convolutional autoencoder + k-means clustering (model I) for the development of signal type model (aI). Top left (a) shows training and validation loss, middle (b) shows the delta label, right (c) shows performance metrics accuracy and F1 for rest evaluated with and without confidence level. Y-axis has the same range as Figure 7.

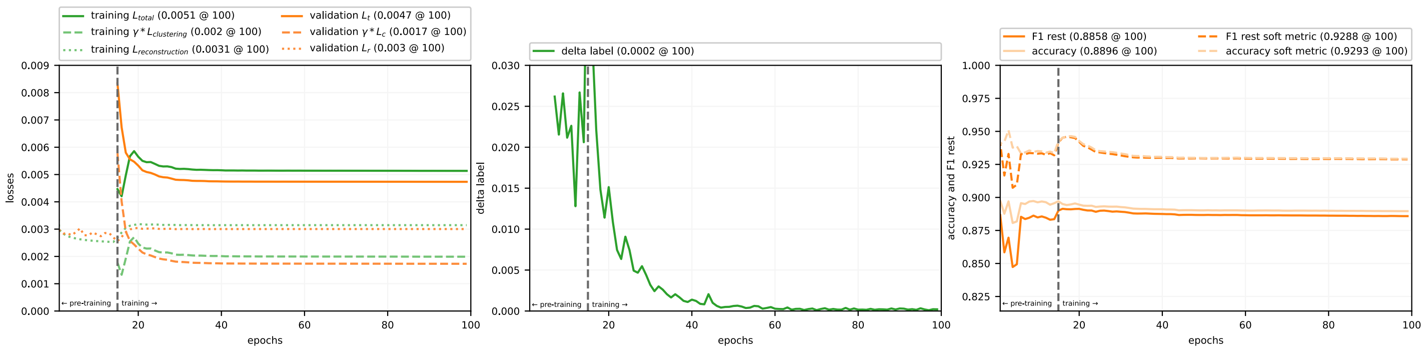


Figure 7: Training and evaluation results of deep convolutional embedded clustering (model II) for the development of signal type model (aII). Top left (a) shows training and validation loss, middle (b) shows the delta label, right (c) shows performance metrics accuracy and F1 for rest evaluated with and without confidence level. The dotted vertical line illustrates the difference between pretraining (left side of the dotted line) and training (right side of the dotted line). Y-axis has the same range as Figure 6.

[2]Master Thesis Deborah Hubers, april 2022

5.1.2 Dataset b: Rest

Following the annotation rules and pre-processing steps resulted in 767 rest Mel spectrograms, 216 fibrillation Mel spectrograms, 388 PSW Mel spectrograms, 9 myotonic discharge Mel spectrograms, 272 CRD Mel spectrograms and 163 fibrillation + PSW Mel spectrograms. Examples of the resultant Mel spectrograms of each class are shown in Appendix D.2.

5.2 Signal Type Model

5.2.1 Hyperparameter Optimisation

The best combination of hyperparameters after performing hyperparameter optimisation for model I and II are shown in Table 1. The search space for the number of features was expanded based on the preliminary results of model I, using the following ranges: the first 500 trials used a range up to 32 features, the second 500 trials used a range up to 64 features, and the final 500 trials used a range up to 4096 features. Further evaluation of the hyperparameter optimisation can be found in Appendix E, including the top 20 results for both models (I and II) and box plots illustrating the relationship between performance and specific hyperparameters (e.g. F1 score for rest vs. number of layers, F1 score for rest vs. batch size).

5.2.2 Training and Evaluation

Model I was trained with the best combination of hyperparameters for 100 epochs. The results are shown in Figure 6. Figure 6a shows the training and validation loss, with the training loss at 0.0031 and the validation loss at 0.0024 after training for 100 epochs. The validation loss remains consistently lower than the training loss throughout training. Figure 6b shows the delta label, which ends at 0.0008 at 100 epochs. This means that 0.08% of the training data is clustered differently compared to the previous epoch. Figure 6c shows the performance metrics accuracy and F1 score for rest, evaluated on the validation set. The F1 score for rest and accuracy stabilises after 25 epochs and after training for 100 epochs they end at 91.8% and 92.0%, respectively. Applying the confidence level results in a higher performance with a final F1 rest value of 98.6% and accuracy of 98.1%. The confidence level was determined to be 53.3% at epoch 100, meaning that all data samples had to belong to their assigned class label (rest, contraction or needle) with minimally 53.3% certainty. The confidence level led to a removal of 20.3% data samples from the validation dataset.

Model II was trained with one adjustment to the selection of best combination of hyperparameters. The selected hyperparameter for the learning rate schedule was the cyclical learning rate schedule without amplitude scaling (clr1), this was changed to the cyclical learning rate schedule with amplitude scaling (clr2). The former schedule did not lead to convergence of the model during training, seen by the fluctuating delta label and performance metrics that do not stabilise, these results are shown in Figure G.1. This may be a result of the select learning rate schedule, because high learning rates (the maximal learning rate is not reduced in clr1) can lead to divergence.

Table 2: Performance matrix of convolutional autoencoder + k-means clustering, evaluated on the test set (from labelled dataset a). In brackets the performance values with confidence are shown. The confidence level for model I was determined to be 53.3%.

	Precision	Recall	F1-score	Support
Rest	0.924 (0.980)	0.847 (0.890)	0.884 (0.932)	5,355 (4,154)
Contraction	0.892 (0.899)	0.899 (0.918)	0.895 (0.909)	5,355 (3,490)
Needle	0.883 (0.928)	0.950 (0.985)	0.916 (0.956)	5,355 (5,030)
Accuracy			0.899 (0.935)	16,065 (12,674)

Table 3: Performance matrix of deep convolutional embedded clustering, evaluated on the test set (from labelled dataset a). In brackets the performance values with confidence are shown. The confidence level for model II was determined to be 97.9%.

	Precision	Recall	F1-score	Support
Rest	0.892 (0.936)	0.857 (0.958)	0.874 (0.947)	5,355 (4,298)
Contraction	0.900 (0.961)	0.828 (0.889)	0.863 (0.924)	5,355 (3,947)
Needle	0.838 (0.904)	0.937 (0.948)	0.885 (0.925)	5,355 (3,856)
Accuracy			0.874 (0.932)	16,065 (12,101)

Model II was then trained with the best combination of hyperparameters and clr2 as learning rate schedule for 100 epochs. The results are shown in Figure 7. Figure 7a shows the training and validation loss, with the combined total training loss at 0.0051 and the combined total validation loss at 0.0047 after training for 100 epochs. The validation loss remains, again, consistently lower than the training loss throughout training. Figure 7b shows the delta label, which ends at 0.0002 at 100 epochs. This means that 0.02% of the training data is clustered differently compared to the previous epoch. Figure 7c shows the performance metrics accuracy and F1 score for rest, evaluated on the validation set. The final F1 score for erest and accuracy at epoch 100 is 88.6% and 89.0%, respectively. Applying the confidence level results in a higher performance with a final F1 rest value of 92.9% and accuracy of 92.9%. The confidence level was determined to be 97.9% at epoch 100, meaning that all data samples had to belong to their assigned class label (rest, contraction or needle) with minimally 97.9% certainty. The confidence level led to a removal of 19.7% data samples from the validation dataset.

5.2.3 Testing

Final model I (epoch 100) was evaluated with the test dataset. Results are shown in Tables 2 and 4. The resultant accuracy on the test set is 89.9% with all data samples and 93.5% after applying the confidence level. Final model II (epoch 100) was evaluated with the test dataset, as well. Results are shown in tables 3 and 5. The resultant accuracy on the test set is 87.4% with all data samples and 93.2% after applying the confidence level.

The features learned by model I (64 features) and by model II (56 features) were reduced to two features by t-SNE. These results are shown in Figure 8 for model I (a) and model II (b). The data samples from test dataset are depicted that were classified with confidence level.

In the appendix are additionally example predictions shown of both models (appendix G.3). It is shown that both models produce visually alike reconstructed images from the input images. The main difference between the input and reconstructed images is that noisy/greyish areas become black in the reconstructed output.

5.3 Rest model

5.3.1 Classification of Mel Spectrograms as Rest

Mel spectrograms from training datasets were used to train the rest model with. Model I classified 374,412 Mel spectrograms and with confidence level 237,158 Mel spectrograms. Model II classified 406,493 Mel spectrograms and with confidence level 315,179 Mel spectrograms. Both datasets with Mel spectrograms classified with confidence level were artificially increased to 500,000 Mel spectrograms.

Mel spectrograms from rest validation set were also classified by both models, and they were only used as validation set when they were classified with the confidence level. These classification results are shown in Table G.1 for model I and in Table G.2 in the appendix. 56% of Mel spectrograms from rest dataset were accurately classified by model I and 52% were accurately classified by model II. The Mel spectrograms were artificially increased to 2,000 Mel spectrograms per class, except for Mel spectrograms from myotonic discharge class.

Table 4: Confusion matrix of convolutional autoencoder + k-means clustering, evaluated on the test set (from labelled dataset a). In brackets the performance values with confidence are shown. The confidence level for model I was determined to be 53.5%.

		Predicted					
		cluster 1	cluster 2	cluster 3	cluster 4	cluster 5	cluster 6
True	Rest	**4535 (3695)**	552 (333)	0 (0)	104 (3)	26 (25)	138 (98)
	Contraction	112 (3))	**3340 (1986)**	1 (1)	0 (0)	**1472 (1217)**	430 (282)
	Needle	260 (74)	6 (0)	**2457 (2457)**	**1986 (1872)**	0 (0)	**646 (627)**

Table 5: Confusion matrix of deep convolutional embedded clustering, evaluated on the test set (from labelled dataset a). In brackets the performance values with confidence are shown. The confidence level for model II was determined to be 97.9%.

		Predicted						
		cluster 1	cluster 2	cluster 3	cluster 4	cluster 5	cluster 6	cluster 7
True	Rest	**3333 (3139)**	441 (118)	182 (10)	0 (0)	47 (25)	**1258 (980)**	94 (26)
	Contraction	45 (25))	**2572 (1734)**	129 (5)	0 (0)	**1863 (1776)**	179 (58)	567 (349)
	Needle	7 (6)	2 (0)	**2216 (1258)**	**2025 (1863)**	2 (1)	325 (194)	**778 (534)**

5.3.2 Training and Evaluation

The rest models (model I and II) were trained with the rest Mel spectrograms for 100 epochs. The results during training are shown in Figure 9 and 10 for model I and II, respectively. The delta label was not always computed for model I, this is because it is required for the computation of the delta label that the cluster centres can be compared from the current epoch with the previous epoch. In case of ambiguity, it is not possible to re-order the cluster numbering and then it was not possible to compute delta label.

The final training loss for model I is 0.0025 and the final validation loss is 0.0031. Delta label is 0.005 at epoch 100. The final accuracy after training is 2.76%, it is seen in the figure that the accuracy fluctuates during training, reaching a maximal accuracy of 30.0% at epoch 68. The final training loss for model II is 0.0061 and the final validation loss is 0.0076. Delta label is 0.0003 at epoch 100. The final accuracy is 12.3% after training, and from the figure we can see that the model II does converge. The results of F1 scores during training are shown in Figure G.2 for model I and in Figure G.3 for model II. For model I it can be seen that the F1 scores fluctuate a lot during training, too. The values for model II stabilise after pre-training and the F1 score for fibrillation class is highest with a final value of 38.5%.

5.3.3 Testing

The final models were not evaluated to compute the performance metrics and the confusion matrix.

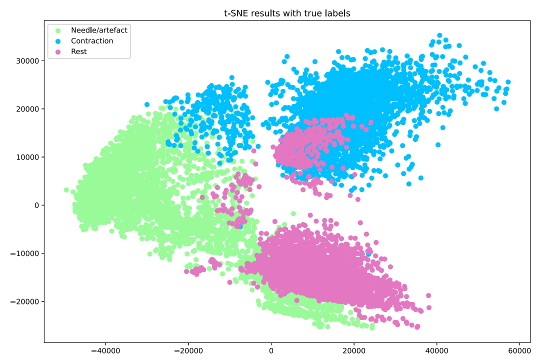
6 Discussion

In this study, we implemented two unsupervised learning strategies for the automatic classification of needle EMG signals. Our results show that both methods (CAE + k-means clustering and DCEC) were able to accurately classify rest, contraction, and needle EMG signals, with a final accuracy of 93.5% on the test set using the confidence level. The classification of phenomena present in rest, such as fibrillation potentials and positive sharp waves, was less successful and requires further research.

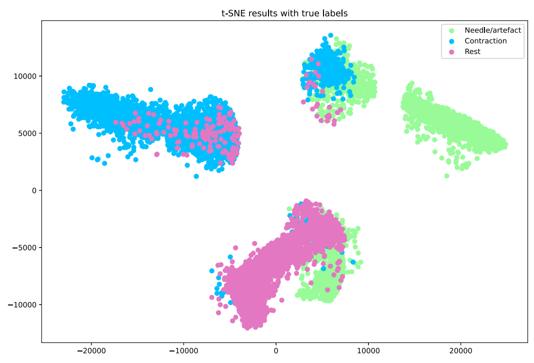
6.1 Signal Type Model

The unsupervised learning models CAE + k-means clustering and DCEC classifies resting state, contraction and needle movements with an accuracy of 93.5% (model I, confidence level 53.3%) and 93.2% (model II, confidence level 97.9%) evaluated on the test set. The confidence level ensures a higher performance at the cost of removing 21.1% of the test set for model I (Table 2) and 24.7% of the test set for model II (Table 3). Model I is best in identifying needle signals from rest and contraction signals with a F1 score of 95.6%. Model II is best in identifying rest signals with a F1 score of 94.7%.

There is no previous literature on the classification of needle EMG signals into different types to compare our results with. However, there are studies on the classification of muscle rest and muscle contraction using surface EMG data. Surface EMG is non-invasive and used for the control of EMG-based prosthetic limbs because it is free of discomfort and with minimal risk of infection [32]. Al-Maliki et al. successfully implemented a stacked autoencoder for the classification of hand movements resulting in a classification accuracy of 99% for the classification of 10 movements, including both resting and contracting state signals [33]. Zia ur Rehman et al. showed that stacked autoencoders performed better than linear discriminant analysis (LDA) [34]. Wang et al. showed that feature extraction using a sparse autoencoder and consequently classification with deep neural network results in successful classification results for the classification of hand movement gestures [35]. A stacked autoencoder is a fully connected neural network that requires more training time and data because there is a higher number of trainable weights compared to a CAE.



(a)



(b)

Figure 8: t-SNE results of model I (a) and model II (b) for test data samples classified with confidence level.

The strength of the classification of rest, contraction and needle with unsupervised learning techniques is that we are not dependent on labelled data. There are limitations to using labelled dataset, such as potential for user bias and differences between annotators, which can introduce selection bias. In the labelled signal type dataset, 24.8% had to be excluded due to unknown or differently labelled data parts. This can be problematic because it may exclude complex data that could have helped the model better discriminate between the classes. Despite these limitations, the labelled dataset can still be useful for evaluating the performance of an unsupervised model, as it can provide a sense of the model's ability to identify strictly separated classes when trained with complex and potentially ambiguous data.

It is, however, better to perform validation with an external dataset (derived from a different hospital and annotated by different people) to compute a more accurately performance. The labelled data set for rest provides additional information on the true performance of the signal type models. The F1 score for rest samples was 94.7% for model II when classifying signals with a confidence level, this indicates that the model is sufficiently able to classify rest signals as rest. However, only 52% of the rest labelled dataset is classified as rest by the signal type DCEC model (Table G.2 in appendix). This dataset is an over representation of spontaneous activity and this data was especially difficult to annotate (represented by the poor interrater reliability of 0.03 average in the files containing spontaneous activity, Appendix F). Two types of spontaneous activity (CRD and myotonic discharge) were also not annotated (rule 9, Appendix A.2).

6.2 Rest Model

The unsupervised learning models CAE + k-means clustering and DCEC were not able to classify spontaneous activity. Model II converged to a final accuracy of 12.3% and model I did not converge. Model II was best in identifying fibrillation potentials from the other classes with a final F1 score of 38.5%.

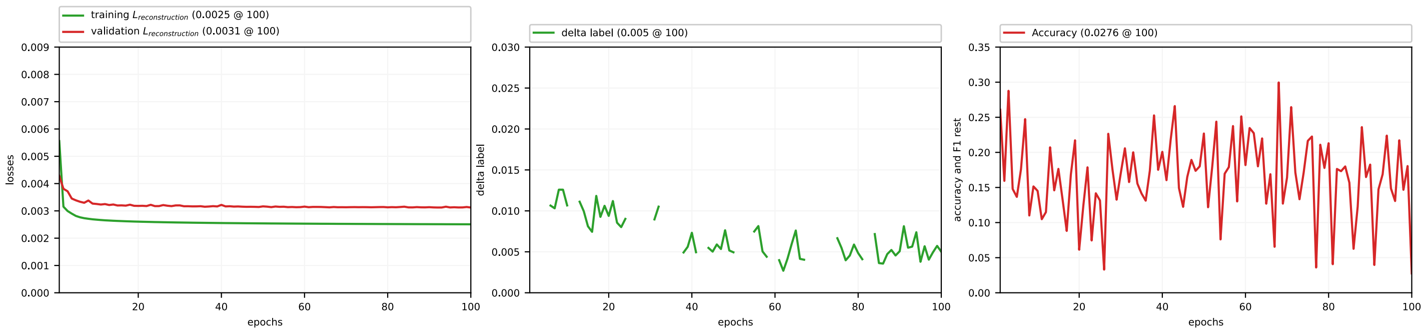


Figure 9: Training and evaluation results of convolutional autoencoder + k-means clustering (model I) for the development of signal type model (aI). Top left (a) shows training and validation loss, middle (b) shows the delta label, right (c) shows performance metrics accuracy and F1 for rest evaluated with and without confidence level. Y-axis has the same range as Figure 9.

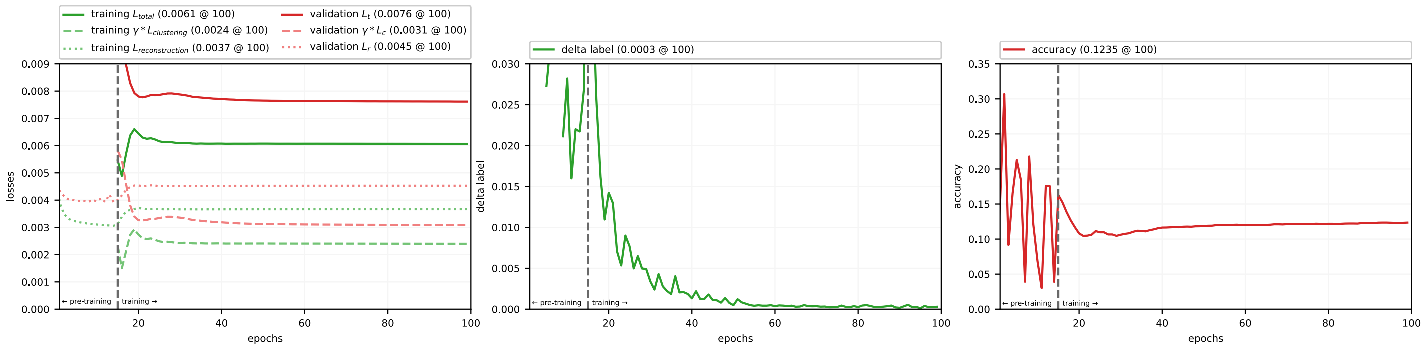


Figure 10: Training and evaluation results of deep convolutional embedded clustering (model II) for the development of signal type model (aII). Top left (a) shows training and validation loss, middle (b) shows the delta label, right (c) shows performance metrics accuracy and F1 for rest evaluated with and without confidence level. The dotted vertical line illustrates the difference between pretraining (left side of the dotted line) and training (right side of the dotted line). Y-axis has the same range as Figure 8.

Nodera et al. have applied deep learning techniques for the classification of rest needle EMG signals [11]. They successfully classified six phenomena present in rest (i.e.: CRD, endplate potentials, fasciculation potentials, fibrillation potentials/PSW, and myotonic discharges) in a supervised classification task after transforming the signals to Mel spectrograms and applying data-augmentation [11]. Training with the original training set of 271 spectrograms resulted in an accuracy of 86% on the test set (containing 59 spectrograms). Training with data-augmentation up to 200,000 training images showed an increase in performance to 100% accuracy on the test set. Their labelled dataset contained two additional classes (endplate potentials and fasciculation potentials) that were not present in the dataset used in the current study. The low classification accuracy obtained in the current study compared to Nodera et al. is most likely due to the use of different training approaches (supervised vs. unsupervised learning) but it may also be due to selection bias in the dataset used by Nodera et al.

The labelled dataset that was used to validate the rest models with, is small. This is a limitation because it is then less representative for each class. Myotonic discharge class, for example, contained 9 Mel spectrograms. Other spontaneous activity classes did not exist at all (for example fasciculation potentials and endplate noise/spikes). Myotonic discharge and fasciculation potentials are rare, and the files that were annotated in our datasets did not contain (many of) these signals. Endplate potentials are normal and could be observed in every patient, they are, however, painful for the patient which is why it is prevented to record these signals for a longer period of time.

6.3 Unsupervised Learning Strategies

We expected that DCEC would outperform CAE + k-means clustering because the features learned by DCEC are optimised by both the clustering loss and the reconstruction loss. However, the testing results showed that model I (CAE + k-means) performed slightly better than model II (DCEC) in classifying rest, contraction, and needle signals. DCEC model, however, seems to generalise better. The accuracy computed on the validation datasets is 92.3% for model II compared to 98.1% for model I, a substantial difference. The difference is however not as noticeable anymore in the testing performance: 93.2% vs. 93.5%. Model I may be more prone to overfitting, the validation data was extensively used to select the best combination of hyperparameters. Additionally, model II resulted in more stable results with the rest data set and the performance was not a random guess in the loss landscape, leading to better performance in the rest model.

Unsupervised learning has been applied in the classification of medical signals in a few number of studies. For example, Wen et al. used a CAE to learn features from EEG signals for the classification of healthy and epilepsy EEG signals. The input for the CAE was a one-dimensional time signal of 4096 samples. The features learned by the CAE were then successfully used in a supervised machine learning classification task [36]. Similarly, Jang et al. proposed an unsupervised feature learning method using a convolutional variational autoencoder for the detection of anomalies in ECG signals [37]. In this study, the input for the autoencoder was a one-dimensional ECG signal containing 2048 time points.

6.4 Limitations and Recommendations

One potential limitation of this work is the decision to convert raw needle EMG signals into Mel spectrograms as input. We made this decision to benefit from the advances in deep learning for image classification and this approach has been successful in previous supervised classification tasks [11, 38]. The current implementation of the Mel spectrogram, however, relies on the 1.48 s time signal (see appendix D.4). This is a drawback because spontaneous activity waveforms are usually of smaller amplitude compared to MUAP waveforms and this information is partly lost. An alternative approach was proposed, but did not result in satisfactory classification performances in early evaluations. To improve the performance of the model in detecting subtle differences between classes, it is useful to focus on optimising the computation of the input data in future work.

Another limitation of the Mel spectrograms as input data is that is more difficult to interpret the model's results. It is desired that it is possible to evaluate the model's prediction so that the model may be used as decision support system. It is possible to provide the needle EMG signals as time signal in convolutional models, this was performed in previous studies where time signals of 4096 and 2048 samples were provided as input [36, 37]. The needle EMG signal, however, is derived with a very high sample frequency and it is thus most likely not feasible to apply this approach.

The current unsupervised learning approach is not effective for classifying spontaneous activity. In order to improve classification performance, it may be useful to adopt a semi-supervised approach [39–42]. Semi-supervised learning allows for the incorporation of both labelled and unlabelled data, which can provide guidance to the model while still benefit from using a large amount of data. One possible approach for implementing a semi-supervised learning strategy is to implement an additional loss function that compares the model's clustering output on a small set of labelled training data, encouraging the model to learn discriminative features between different classes. It is important to carefully consider which type of semi-supervised approach would be most suitable for our specific use case, as discussed in [41].

During needle EMG examination, the time course of the signal is important for evaluating the presence of abnormal spontaneous activity. Normal spontaneous activity may occur immediately after needle insertion, but prolonged spontaneous activity at rest is abnormal. To accurately classify these signals, it is important for the model to consider the sequential nature of the data. One type of deep learning model that can handle sequential data is a long short-term memory (LSTM) network. LSTM networks were successfully implemented in the classification of hand gestures using surface EMG signals in several studies [43–45].

The aim of this study was to classify spontaneous activity from hospital-acquired needle EMG signals. In future research, it would be interesting to apply the same approach to contraction signals as well. The ultimate goal is to classify the entire hospital-acquired needle EMG signal, which requires the ability to identify MUAP waveforms. In particular, it is desired to distinguish between myopathic and neuropathic MUAPs [2].

7 Conclusion

In this paper, we presented the first application of unsupervised learning techniques for the automatic classification of needle EMG signals. By training our models on a large amount of data from our clinical database, we were able to take advantage of deep learning and eliminate user-dependent bias and interrater differences in data annotation. Additionally, our use of raw data rather than manually selected samples helps to prevent selection bias. The signal type models were able to classify spontaneous, contraction, and insertional activity, and this model could be used as a preprocessing step in the classification of hospital-acquired needle EMG data to avoid manual preprocessing and increase the number of available data samples. Our study provides valuable insights for the use of unsupervised learning in the automatic classification of needle EMG signals and highlights the need for further research in this area.

Bibliography

[1] Devon I. Rubin. Needle electromyography: Basic concepts. In *Handbook of Clinical Neurology*, volume 160, pages 243–256. Elsevier B.V., 1 2019. doi: 10.1016/B978-0-444-64032-1.00016-3. URL https://linkinghub.elsevier.com/retrieve/pii/B9780444640321000163.

[2] David C. Preston and Barbara Ellen Shapiro. *Electromyography and Neuromuscular Disorders*, volume 4th edition. 2020. ISBN 9780323758291.

[3] Jee-Eun Kim, Jin Myoung Seok, Suk-Won Ahn, Byung-Nam Yoon, Young-Min Lim, Kwang-Kuk Kim, Ki-Han Kwon, Kee Duk Park, and Bum Chun Suh. Basic concepts of needle electromyography. *Annals of Clinical Neurophysiology*, 21(1):7, 2019. ISSN 2508-691X. doi: 10.14253/acn.2019.21.1.7. URL https://synapse.koreamed.org/DOIx.php?id=10.14253/acn.2019.21.1.7.

[4] Pushpa Narayanaswami, Thomas Geisbush, Lyell Jones, Michael Weiss, Tahseen Mozaffar, Gary Gronseth, and Seward B. Rutkove. Critically re-evaluating a common technique. *Neurology*, 86(3):218–223, 1 2016. ISSN 0028-3878. doi: 10.1212/WNL.0000000000002292.

[5] Richard Kendall and Robert A. Werner. Interrater reliability of the needle examination in lumbosacral radiculopathy. *Muscle & Nerve*, 34(2):238–241, 8 2006. ISSN 0148-639X. doi: 10.1002/mus.20554. URL https://onlinelibrary.wiley.com/doi/10.1002/mus.20554.

[6] Jonathan R. Torres-Castillo, Carlos Omar López-López, and Miguel A. Padilla-Castañeda. Neuromuscular disorders detection through time-frequency analysis and classification of multi-muscular EMG signals using Hilbert-Huang transform. *Biomedical Signal Processing and Control*, 71:103037, 1 2022. ISSN 17468094. doi: 10.1016/j.bspc.2021.103037. URL https://linkinghub.elsevier.com/retrieve/pii/S1746809421006340.

[7] Rohit Bose, Kaniska Samanta, Sudip Modak, and Soumya Chatterjee. Augmenting Neuromuscular Disease Detection Using Optimally Parameterized Weighted Visibility Graph. *IEEE Journal of Biomedical and Health Informatics*, 25(3):685–692, 3 2021. ISSN 21682208. doi: 10.1109/JBHI.2020.3001877.

[8] Kaniska Samanta, Sayanjit Singha Roy, Sudip Modak, Soumya Chatterjee, and Rohit Bose. Neuromuscular Disease Detection Employing Deep Feature Extraction from Cross Spectrum Images of Electromyography Signals. In *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pages 694–697. IEEE, 7 2020. ISBN 978-1-7281-1990-8. doi: 10.1109/EMBC44109.2020.9176464. URL https://ieeexplore.ieee.org/document/9176464/.

[9] Tahereh Kamali and Daniel W. Stashuk. Transparent Electrophysiological Muscle Classification from EMG Signals Using Fuzzy-Based Multiple Instance Learning. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 28(4):842–849, 4 2020. ISSN 15580210. doi: 10.1109/TNSRE.2020.2979412.

[10] Abdulkadir Sengur, Yaman Akbulut, Yanhui Guo, and Varun Bajaj. Classification of amyotrophic lateral sclerosis disease based on convolutional neural network and reinforcement sample learning algorithm. *Health Information Science and Systems*, 5(1), 12 2017. ISSN 2047-2501. doi: 10.1007/s13755-017-0029-6.

[11] Hiroyuki Nodera, Yusuke Osaki, Hiroki Yamazaki, Atsuko Mori, Yuishin Izumi, and Ryuji Kaji. Deep learning for waveform identification of resting needle electromyography signals. *Clinical Neurophysiology*, 130(5):617–623, 5 2019. ISSN 18728952. doi: 10.1016/j.clinph.2019.01.024.

[12] Tien En Chen, Shih I. Yang, Li Ting Ho, Kun Hsi Tsai, Yu Hsuan Chen, Yun Fan Chang, Ying Hui Lai, Syu Siang Wang, Yu Tsao, and Chau Chung Wu. S1 and S2 heart sound recognition using deep neural networks. *IEEE Transactions on Biomedical Engineering*, 64(2):372–380, 2 2017. ISSN 15582531. doi: 10.1109/TBME.2016.2559800.

[13] Roman A. Solovyev, Maxim Vakhrushev, Alexander Radionov, Irina I. Romanova, Aleksandr A. Amerikanov, Vladimir Aliev, and Alexey A. Shvets. Deep Learning Approaches for Understanding Simple Speech Commands. In *2020 IEEE 40th International Conference on Electronics and Nanotechnology, ELNANO 2020 - Proceedings*, pages 688–693. Institute of Electrical and Electronics Engineers Inc., 4 2020. ISBN 9781728197135. doi: 10.1109/ELNANO50318.2020.9088863.

[14] Mohammed Aly and Nouf Saeed Alotaibi. A novel deep learning model to detect COVID-19 based on wavelet features extracted from Mel-scale spectrogram of patients' cough and breathing sounds. *Informatics in Medicine Unlocked*, 32, 1 2022. ISSN 23529148. doi: 10.1016/j.imu.2022.101049.

[15] M Sc Miki Nikolic, Christian Krarup, and John Aasted Sørensen. Detailed Analysis of Clinical Electromyography Signals. Technical report, University of Copenhagen, Copenhagen, 8 2001.

[16] Michael Steinbach, Levent Ertöz, and Vipin Kumar. The Challenges of Clustering High Dimensional Data. In *New Directions in Statistical Physics*, pages 273–309. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. doi: 10.1007/978-3-662-08968-2{_}16. URL http://link.springer.com/10.1007/978-3-662-08968-2_16.

[17] Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised Deep Embedding for Clustering Analysis. 11 2015. URL http://arxiv.org/abs/1511.06335.

[18] Xifeng Guo, Xinwang Liu, En Zhu, and Jianping Yin. Deep Clustering with Convolutional Autoencoders. pages 373–382. 2017. doi: 10.1007/978-3-319-70096-0{_}39. URL http://link.springer.com/10.1007/978-3-319-70096-0_39.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[20] Mohammad Hesam Hesamian, Wenjing Jia, Xiangjian He, and Paul Kennedy. Deep Learning Techniques for Medical Image Segmentation: Achievements and Challenges. *Journal of Digital Imaging*, 32(4):582–596, 8 2019. ISSN 1618727X. doi: 10.1007/s10278-019-00227-x.

[21] Samir S. Yadav and Shivajirao M. Jadhav. Deep convolutional neural network based medical image classification for disease diagnosis. *Journal of Big Data*, 6(1), 12 2019. ISSN 21961115. doi: 10.1186/s40537-019-0276-2.

[22] Lei Cai, Jingyang Gao, and Di Zhao. A review of the application of deep learning in medical image classification and segmentation. *Annals of Translational Medicine*, 8(11):713–713, 6 2020. ISSN 23055839. doi: 10.21037/atm.2020.02.44. URL http://atm.amegroups.com/article/view/36944/html.

[23] Stuart P. Lloyd. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982. ISSN 15579654. doi: 10.1109/TIT.1982.1056489.

[24] S. S. Stevens, J. Volkmann, and E. B. Newman. A Scale for the Measurement of the Psychological Magnitude Pitch. *The Journal of the Acoustical Society of America*, 8(3):185–190, 1 1937. ISSN 0001-4966. doi: 10.1121/1.1915893.

[25] Christian J Steinmetz and Joshua D Reiss. pyloudnorm: A simple yet flexible loudness meter in Python. In *150th AES Convention*. 150th AES Convention, 150th AES Convention, 5 2021. URL https://ffmpeg.org/.

[26] Brian Mcfee, Colin Raffel, Dawen Liang, Daniel P W Ellis, Matt Mcvicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and Music Signal Analysis in Python. Technical report, 2015. URL https://www.youtube.com/watch?v=MhOdbtPhbLU.

[27] Ekin Yagis, Selamawet Workalemahu Atnafu, Alba García Seco de Herrera, Chiara Marzi, Riccardo Scheda, Marco Giannelli, Carlo Tessa, Luca Citi, and Stefano Diciotti. Effect of data leakage in brain MRI classification using 2D convolutional neural networks. *Scientific Reports*, 11(1):22544, 12 2021. ISSN 2045-2322. doi: 10.1038/s41598-021-01681-w.

[28] F. Chollet and others. Keras. GitHub. , 2015.

[29] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in Python. Technical report, 2011. URL http://scikit-learn.sourceforge.net.

[30] Matt Crooks. Confidence in k-means, 7 2019.

[31] J Bergstra, D Yamins, and D D Cox. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. Technical report, 2013.

[32] Nurhazimah Nazmi, Mohd Azizi Abdul Rahman, Shin Ichiroh Yamamoto, Siti Anom Ahmad, Hair Zamzuri, and Saiful Amri Mazlan. A review of classification techniques of EMG signals during isotonic and isometric contractions, 8 2016. ISSN 14248220.

[33] Abdullah Y. Al-Maliki and Kamran Iqbal. Hand and Lower Arm Movements Classification Using Deep ANN and sEMG. In *IEEE International Symposium on Industrial Electronics*, volume 2021-June. Institute of Electrical and Electronics Engineers Inc., 6 2021. ISBN 9781728190235. doi: 10.1109/ISIE45552.2021.9576387.

[34] Muhammad Zia ur Rehman, Syed Gilani, Asim Waris, Imran Niazi, Gregory Slabaugh, Dario Farina, and Ernest Kamavuako. Stacked Sparse Autoencoders for EMG-Based Classification of Hand Motions: A Comparative Multi Day Analyses between Surface and Intramuscular EMG. *Applied Sciences*, 8(7):1126, 7 2018. ISSN 2076-3417. doi: 10.3390/app8071126. URL http://www.mdpi.com/2076-3417/8/7/1126.

[35] Xiaojie Wang, Yucheng Wang, Zhonghui Wang, chunhui wang, and You Li. Hand gesture recognition using sparse autoencoder-based deep neural network based on electromyography measurements. page 42. SPIE-Intl Soc Optical Eng, 3 2018. ISBN 9781510616905. doi: 10.1117/12.2296382.

[36] Tingxi Wen and Zhongnan Zhang. Deep Convolution Neural Network and Autoencoders-Based Unsupervised Feature Learning of EEG Signals. *IEEE Access*, 6:25399–25410, 2018. ISSN 2169-3536. doi: 10.1109/ACCESS.2018.2833746. URL https://ieeexplore.ieee.org/document/8355473/.

[37] Jong Hwan Jang, Tae Young Kim, Hong Seok Lim, and Dukyong Yoon. Unsupervised feature learning for electrocardiogram data using the convolutional variational autoencoder. *PLoS ONE*, 16(12 December), 12 2021. ISSN 19326203. doi: 10.1371/journal.pone.0260612.

[38] Hiroyuki Nodera, Yusuke Osaki, Hiroki Yamazaki, Atsuko Mori, Yuishin Izumi, and Ryuji Kaji. Classification of needle-EMG resting potentials by machine learning. *Muscle and Nerve*, 59(2):224–228, 2 2019. ISSN 10974598. doi: 10.1002/mus.26363.

[39] Shuai Chen, Gerda Bortsova, Antonio García-Uceda Juárez, Gijs van Tulder, and Marleen de Bruijne. Multi-task Attention-Based Semi-supervised Learning for Medical Image Segmentation. pages 457–465. 2019. doi: 10.1007/978-3-030-32248-9{_}51.

[40] Rushi Jiao, Yichi Zhang, Le Ding, Rong Cai, and Jicong Zhang. Learning with Limited Annotations: A Survey on Deep Semi-Supervised Learning for Medical Image Segmentation. 7 2022. URL http://arxiv.org/abs/2207.14191.

[41] Jesper E. van Engelen and Holger H. Hoos. A survey on semi-supervised learning. *Machine Learning*, 109(2):373–440, 2 2020. ISSN 15730565. doi: 10.1007/s10994-019-05855-6.

[42] I. Zeki Yalniz, Hervé Jégou, Kan Chen, Manohar Paluri, and Dhruv Mahajan. Billion-scale semi-supervised learning for image classification. 5 2019. URL http://arxiv.org/abs/1905.00546.

[43] Pufan Xu, Fei Li, and Haipeng Wang. A novel concatenate feature fusion RCNN architecture for sEMG-based hand gesture recognition. *PLoS ONE*, 17(1 January), 1 2022. ISSN 19326203. doi: 10.1371/journal.pone.0262810.

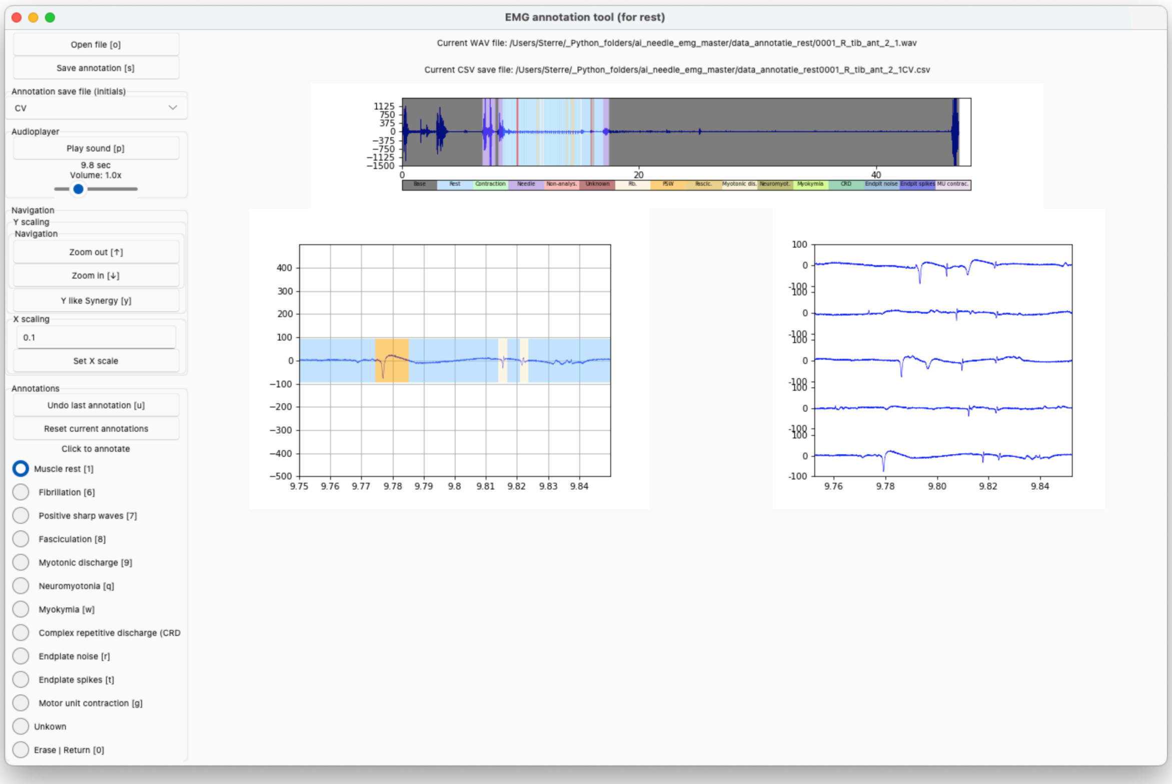
[44] Weidong Geng, Yu Hu, Yongkang Wong, Wentao Wei, Yu Du, and Mohan Kankanhalli. A novel attention-based hybrid CNN-RNN architecture for sEMG-based gesture recognition. *PLoS ONE*, 13(10), 10 2018. ISSN 19326203. doi: 10.1371/journal.pone.0206049.

[45] Dawei Huang and Badong Chen. Surface EMG Decoding for Hand Gestures Based on Spectrogram and CNN-LSTM. In *2019 2nd China Symposium on Cognitive Computing and Hybrid Intelligence (CCHI)*, pages 123–126. IEEE, 9 2019. ISBN 978-1-7281-4091-9. doi: 10.1109/CCHI.2019.8901936. URL https://ieeexplore.ieee.org/document/8901936/.

A Annotation: Rules and Tool

A.1 Annotation Tool: updated for rest signals

The annotation tool that was developed in a previous project (Deborah Hubers, Master thesis, april 2022) was used to make annotation. The tool is written in Python and used Tkinter. The tool was updated to allow for the annotation of phenomena present in rest. Figure A.1 shows a screenshot of the tool, which includes options for opening a .wav file, saving annotations, selecting the person performing the annotation, playing the audio, and adjusting the sound with a sliding bar. The tool also allows for navigation of the main figure (located in the middle left of the tool) on the x- and y-axes and for making annotations by selecting the type of annotation (using either the mouse or a numbered shortcut) and clicking twice on the main figure to mark the start and end of the annotation.



EMG annotation tool (for rest)

Current WAV file: /Users/Sterre/_Python_folders/al_needle_emg_master/data_annotatie_rest/0001_R_tib_ant_2_1.wav

Current CSV save file: /Users/Sterre/_Python_folders/al_needle_emg_master/data_annotatie_rest0001_R_tib_ant_2_1CV.csv

Annotation save file (initials): CV

9.8 sec

Volume: 1.0x

X scaling: 0.1

Figure A.1: The annotation tool that was updated for the annotation of spontaneous activity in rest. the annotations for rest that can be made are shown on the left-hand side under Muscle rest. The red line in the upper figure shows the location of the signal shown in the left middle figure.

A.2 Annotation for Signal Type model

The following rules were applied:

1. Signal rest is annotated as signal rest
2. When the signal was intended as rest but contains a firing moter unit action potential, the signal is annotated as rest.
3. Spontaneous rest activity is annotated as rest.
4. Needle movement is annotated as needle movement.
5. Needle in the air is annotated as non-analysable. (In this model treated the same as needle movement)
6. Contraction is annotated as contraction.
7. Distant contraction is annotated as contraction.
8. Myotonic discharges and complex repetitive discharges are left black (no annotation).
9. When in doubt: no annotation.

A.3 Annotation for Rest model

The following rules were applied:

1. Signal rest that did not contain spontaneous rest activity is annotated as rest.
2. Spontaneous activity (i.e. fibrillation potentials, positive sharp waves, fasciculation potentials, myotonic discharge, neuromyotonia, myokymia, complex repetitive discharge, endplate noise, endplate spikes, motor unit contraction) is annotated as spontaneous activity. If waveforms are separately discernible, they are separate annotated.
3. When in doubt: no annotation.

B Code changes

This Master Thesis is an implementation of a model presented in the paper by Guo et al. [1]. The accompanying code can be found in the GitHub link: `https://github.com/XifengGuo/DCEC`. Innovations in deep learning are often tested in small-scale experiments using standardised databases. The implementation of the code then consists of adapting the code to the specific user case. We encountered, however, additional problems in the implementation of the code, specifically related to the clustering layer. This appendix will discuss the problems and the structural changes that had to be adopted to implement the clustering layer.

The clustering loss is the Kullback-Leibler divergence of a target distribution p and soft label q. The soft label is determined during training (for one batch) where an input is mapped by the clustering layer to a soft label by the Student's t-distribution. The target distribution is subsequently derived over all soft labels (thus over all training samples). In the code listing below (Listing 1) it is shown how this method was embedded in the original code. At the beginning of each epoch, on line 3, a prediction is made using all training data. For the clustering output, this results in soft label q. The target distribution p is consequently derived which is used during training. The 'train_on_batch' method is employed which allows the supply of both targeted outputs. The reconstruction loss (mean squared error loss) is computed over the predicted output by the model and the input sample. The clustering loss (Kullback-Leibler divergence) is computed over the predicted soft label q and the target distribution p.

```
1  for epoch in trange(1, epochs_dcec+1): # loop over all epochs
2      # predict using all training data and derive q
3      q, _ = self.model.predict(x_train, verbose=0)
4      # compute target distribution p
5      weight = q ** 2 / q.sum(0)
6      p = (weight.T / weight.sum(1)).T
7  
8      # train on batch
9      for ite in tqdm(range(iterations)): # iterations = train_data.samples / batch_size (= bs)
10         x_samples = next(iter(train_data))
11         x_samples = x_samples[0]
12 
13         if len(x_samples) < batch_size:
14             loss = self.model.train_on_batch(x=x_samples, y=[p[ite*bs::], x_samples])
15         else:
16             loss = self.model.train_on_batch(x=x_samples, y=[p[ite*bs: (ite+1)*bs], x_samples])
```

Listing 1: Original implementation of training with clustering loss.

The authors thus implemented this method by using Keras 'train_on_batch' method. The following problems were encountered with the implementation of this code:

1. Slow code - This is especially problematic (leading to extra time and extra costs) for performing the hyper-parameter optimisation.
2. Validation loss cannot be computed - Loss can only be derived by using the 'train_on_batch' line (because this allows to provide the targeted output) but this means that the validation loss cannot be computed. The validation loss is an evaluation of the model where no training is involved.
3. Not memory efficient - An out of memory error occurs when predicting the models output using all available training data (1,3 million Mel spectrograms).

The code originates from 2017, which is a different era in terms of coding possibilities. The 'train_on_batch' method was at the time an accessible solution to deal with different inputs, outputs and customise how losses need to be computed. This method is however not desired because it is the main reason for the slow code because the GPU is not accessed efficiently. Since 2020 it is possible to customise what happens in the 'fit' function (the general function to use for training deep learning methods with Keras) by overriding the 'train_step' function. This function is called by 'fit' and for each batch the loss is calculated and the weights are consequently updated. The overridden version of the 'train_step' function is shown in Listing 2.

```
def train_step(self, data):
    # Override method 'train_step' to customize how training takes place.
    with tf.GradientTape() as tape:
        data, p = data[0], data[1] # input data in batches
        reconstruction, q = self(data) # forward pass
        # compute losses
        mse = tf.keras.losses.MeanSquaredError()
        reconstruction_loss = mse(data,reconstruction)
        kld = tf.keras.losses.KLDivergence()
        kl_loss = kld(p, q)
```

```
11          # compute total loss
12          total_loss = reconstruction_loss + kl_loss * self.gamma
13
14      # compute gradients with total loss
15      grads = tape.gradient(total_loss, self.trainable_weights)
16      # update weights
17      self.optimizer.apply_gradients(zip(grads, self.trainable_weights))
18      # update metrics
19      self.total_loss_tracker.update_state(total_loss)
20      self.reconstruction_loss_tracker.update_state(reconstruction_loss)
21      self.kl_loss_tracker.update_state(kl_loss)
22
23      # return a dict mapping metric names to current value
24      return {
25          "loss": self.total_loss_tracker.result(),
26          "reconstruction_loss": self.reconstruction_loss_tracker.result(),
27          "clustering_loss": self.kl_loss_tracker.result(),
28      }
```

Listing 2: Implementation of how to customise what happens in fit function.

The data (in batches) is unpacked on line 4. What is shown here, is that parameter p is also given as input. The target distribution is computed at the beginning of an epoch using all training data. The target distribution p was joined with the input data (the Mel spectrograms). The implementation of the code is shown in Listing 3.

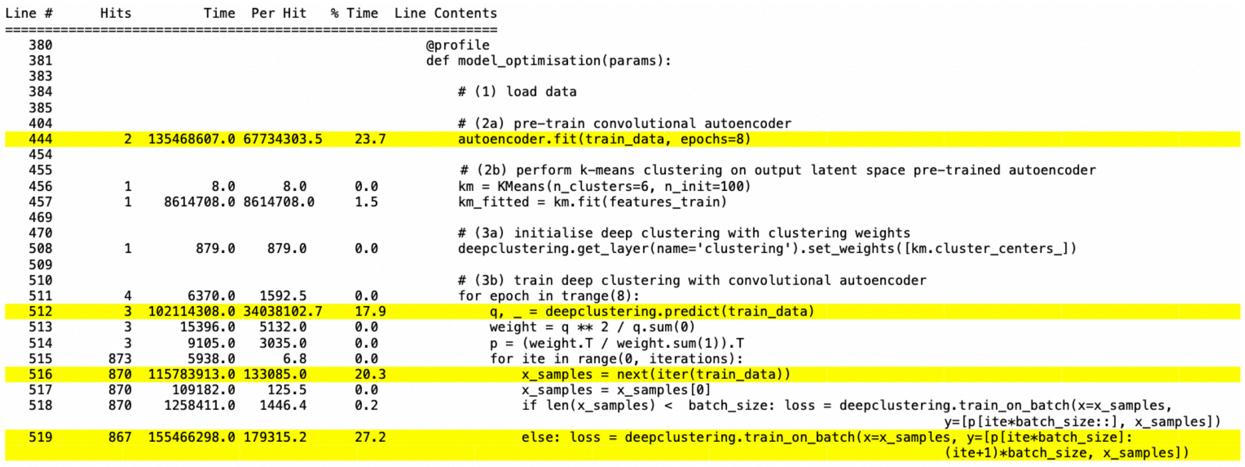
```
1  for epoch in range(1, epochs_dcec+1): # loop over all epochs
2
3      # define separate instance of model with only clustering layer as output
4      clustering = Model(inputs=model.input, outputs=model.output[1])
5
6      q = clustering.predict(train_data_dcec)
7      y_pred = q.argmax(1)
8      weight = q ** 2 / q.sum(0)
9      p = (weight.T / weight.sum(1)).T
10
11     combined_input = JoinedGen(train_data_dcec, p, batch_size)
12
13     loss = model.fit(x=combined_input, epochs=1, callbacks=[Callback_DCEC(train_data=
       train_data_dcec, val_data=val_data, y_true_val=y_true_val, labels=labels_val, n_clusters=
       clusters, save_directory=config.save_directory, batch_size=batch_size, epoch=epoch,
       y_pred_last=y_pred)])
```

Listing 3: Implementation of how to customise what happens in fit function.

It is not possible to call fit for multiple epochs, because the target distribution p needs to be computed after each epoch. It is thus still necessary to loop over all epochs. In line 4 we define a separate model that only consists of the encoder, the latent space and the clustering output. Using this model to predict we only receive the output of the clustering layer, q, and not the reconstructed image as output. As a result, we do not get an out of memory error when predicting with all the training data (1.3 million Mel spectrograms).

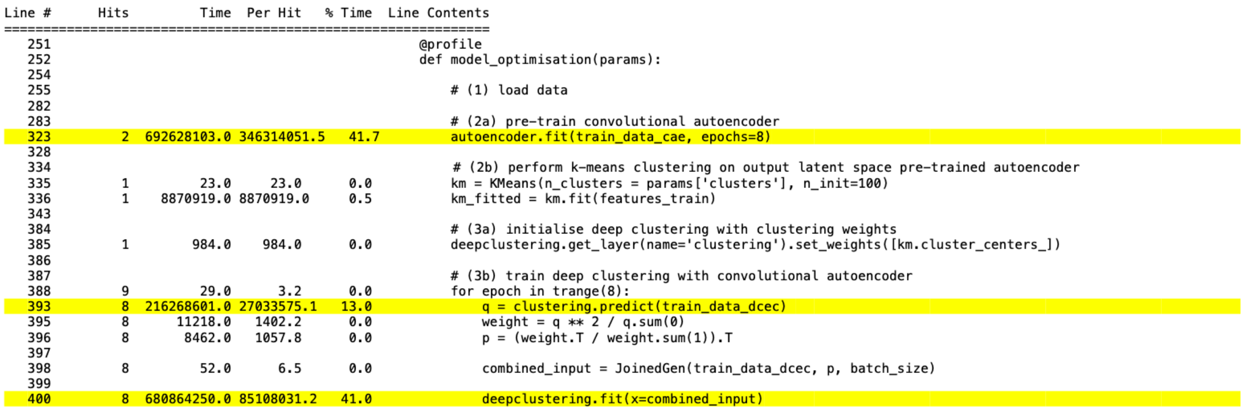
We measured the difference in execution time between the original code and the updated code using 'line-profiler'. This profiles the time individual lines of code take to execute The profiler was implemented to evaluate execution time during hyperparameter optimisation. Hyperparameter optimisation for DCEC consists of a pretraining stage (training of convolutional autoencoder) and training stage (training of convolutional autoencoder + clustering layer). After pretraining, the cluster centres of the clustering layer are initialised by performing k-means on learned features of all input data.

The results of the profiler are shown in Figure B.1. We can use the % time units to calculate the efficiency gain between the original implementation of the code (Fig. B.1a) and the updated implementation of the code (Fig. B.1b). We can see that the relative difference in time has decreased for step 2 vs step 3 after updating the code and we have an efficiency gain of 50% for step 3.



```
Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   380                                           @profile
   381                                           def model_optimisation(params):
   383
   384                                               # (1) load data
   385
   404                                               # (2a) pre-train convolutional autoencoder
   444         2  135468607.0 67734303.5     23.7      autoencoder.fit(train_data, epochs=8)
   454
   455                                               # (2b) perform k-means clustering on output latent space pre-trained autoencoder
   456         1          8.0        8.0      0.0      km = KMeans(n_clusters=6, n_init=100)
   457         1    8614708.0  8614708.0      1.5      km_fitted = km.fit(features_train)
   469
   470                                               # (3a) initialise deep clustering with clustering weights
   508         1        879.0      879.0      0.0      deepclustering.get_layer(name='clustering').set_weights([km.cluster_centers_])
   509
   510                                               # (3b) train deep clustering with convolutional autoencoder
   511         4       6370.0     1592.5      0.0      for epoch in trange(8):
   512         3  102114308.0 34038102.7     17.9          q, _ = deepclustering.predict(train_data)
   513         3      15396.0     5132.0      0.0          weight = q ** 2 / q.sum(0)
   514         3       9105.0     3035.0      0.0          p = (weight.T / weight.sum(1)).T
   515       873       5938.0        6.8      0.0          for ite in range(0, iterations):
   516       870  115783913.0   133085.0     20.3              x_samples = next(iter(train_data))
   517       870     109182.0      125.5      0.0              x_samples = x_samples[0]
   518       870    1258411.0     1446.4      0.2              if len(x_samples) <  batch_size: loss = deepclustering.train_on_batch(x=x_samples,
                                                                                                   y=[p[ite*batch_size::], x_samples])
   519       867  155466298.0   179315.2     27.2              else: loss = deepclustering.train_on_batch(x=x_samples, y=[p[ite*batch_size]:
                                                                                                   (ite+1)*batch_size, x_samples])
```

(a) Implementation of original code



```
Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   251                                           @profile
   252                                           def model_optimisation(params):
   254
   255                                               # (1) load data
   282
   283                                               # (2a) pre-train convolutional autoencoder
   323         2  692628103.0 346314051.5    41.7      autoencoder.fit(train_data_cae, epochs=8)
   328
   334                                               # (2b) perform k-means clustering on output latent space pre-trained autoencoder
   335         1         23.0       23.0      0.0      km = KMeans(n_clusters = params['clusters'], n_init=100)
   336         1    8870919.0  8870919.0      0.5      km_fitted = km.fit(features_train)
   343
   384                                               # (3a) initialise deep clustering with clustering weights
   385         1        984.0      984.0      0.0      deepclustering.get_layer(name='clustering').set_weights([km.cluster_centers_])
   386
   387                                               # (3b) train deep clustering with convolutional autoencoder
   388         9         29.0        3.2      0.0      for epoch in trange(8):
   393         8  216268601.0 27033575.1     13.0          q = clustering.predict(train_data_dcec)
   395         8      11218.0     1402.2      0.0          weight = q ** 2 / q.sum(0)
   396         8       8462.0     1057.8      0.0          p = (weight.T / weight.sum(1)).T
   397
   398         8         52.0        6.5      0.0          combined_input = JoinedGen(train_data_dcec, p, batch_size)
   399
   400         8  680864250.0 85108031.2     41.0          deepclustering.fit(x=combined_input)
```

(b) Implementation of updated code

Figure B.1: Results of line-profiler to evaluate the execution time of individual lines of code. The function used in hyperparameter optimisation (called model_optimisation) is evaluated. The code is simplified and the parts that are time-consuming are highlighted in yellow.

C Hyperparameter Optimisation: Selection of Parameters

C.1 Introduction

Two models were implemented in this thesis. We performed hyperparameter optimisation to fine-tune these models to our input data. The hyperparameter optimisation tool Hyperopt [2] was implemented to efficiently achieve the best selection of hyperparameters.

The selection of the search space is discussed in the following sections. In Section C.2 the set of hyperparameters are discussed that are important during training. In the subsequent section (Section C.3) the parameters are discussed that make up the models architecture and are hence related to the complexity of the model. The final section (Section C.4) discusses the parameters that can be fine-tuned during clustering. The search space that we implemented is almost the same for both models, where a few additional parameters are added for model two. The final search space is depicted in Table 1 of the main paper.

C.2 Training parameters

C.2.1 Optimisation algorithm

The aim during training is to reach a local optimum in the loss landscape. A deep learning model is in convergence when the loss function reaches a minimum, which can be a global or local minimum. The weights and biases of trainable parameters are updated during training by an optimisation algorithm. Most optimisation algorithms are build upon gradient descent, where the loss function is minimised by following the gradient. One optimisation algorithm was selected for the hyperparameter optimisation. This algorithm is built on other algorithms, stochastic gradient descent, momentum and RMSprop, which will thus also be discussed in the following paragraphs.

Stochastic gradient descent (SGD) (or mini-batch gradient descent) is the simplest optimisation algorithm that employs gradient descent. Exact gradients are computed from a small number of samples (equal to the batch size) to approximate the gradients from all samples. Parameters (θ) are updated by the following formulae, with learning rate $= \alpha$ and objective function $= J$:

$$\theta' = \theta - \alpha \Delta_\theta J(\theta; x^{i:i+n}; y^{i:i+n})$$

Gradient descent is only aware of the (current) gradient and thus only if the loss is declining and how fast. It is however useful "know" more, e.g., what the shape of the loss landscape is, if the current direction is similar to previous updates, etc. Second-order optimisation algorithms provide information about the shape of the loss landscape, such algorithms are however computationally too expensive and therefore not practical for implementation. Past statistics are incorporated in a group of optimisation algorithms using the exponentially weighted moving average. These algorithms can roughly be divided in three groups: (1) Momentum, (2) RMSprop and (3) Adam. Momentum accumulates the gradients of past steps so that the weights are on average updated in the right direction. RMSprop measures the variance and when this is high (indication of the wrong direction) the update will be low. Adam combines both momentum and RMSprop, where the parameters are updated by:

$$\theta' = \theta - \alpha \frac{\hat{v}_i}{\sqrt{\hat{r}_i}}$$

With $\hat{v}_i$ the exponentially weighted moving average for past gradients, calculated by:

$$v_i = p_1 v_{i-1} + (1 - p_1)\Delta_\theta$$
$$\hat{v}_i = \frac{v_i}{1 - p_1^i}$$

And $\hat{r}_i$ is the exponentially weighted moving average for past variance:

$$r_i = p_2 r_{i-1} + (1 - p_2)\Delta_\theta$$
$$\hat{v}_i = \frac{v_i}{1 - p_1^i}$$

C.2.2 Batch size

The batch size denotes the number of images that are seen during one update iteration, and are thus used to compute an approximation of the gradient. A larger batch size may intuitively lead to a better approximation, however, in practice it has been observed that larger batches leads to degradation in the quality of the model (measured by its ability to generalise) [3]. Lack of generalisation may be due to the fact that large-batch methods tend to converge to sharp minimisers of the training function [4]. Typical batch size range lies between 32 and 512 data samples and for this thesis we will test a small and large batch size, respectively 64 and 256.

C.2.3 Learning rate

The learning rate may be the single most important hyperparameter that needs to be tuned for training deep learning models [5, 6]. The learning rate determines the rate of change: a small learning rate converges very slowly while a large learning rate leads to divergence. We explore different learning rate schedules and learning rate values.

A straightforward learning rate schedule is a constant value that remains unchanged during training. The values chosen differs per batch size. A constant learning rate in combination with Adam optimiser is not entirely constant, since Adam also influences the rate of change.

We additionally explore a cyclical learning rate schedule. This schedule is proposed by Smith [7] because he argues that a constant learning rate may converge on a saddle point in the loss landscape (gradients are small but it is not a local minimum). In the cyclical learning schedule the learning rate is cyclically varied within a band of values (see Figure C.1) so that the learning rate alternates between both smaller and larger values. The step size is set at 2, which means that after 2 epochs the learning rate is changed and a total of 4 iterations are needed to complete one cycle. Training is best stopped after the end of a cycle, which is therefore after 4, 8, etc. epochs.

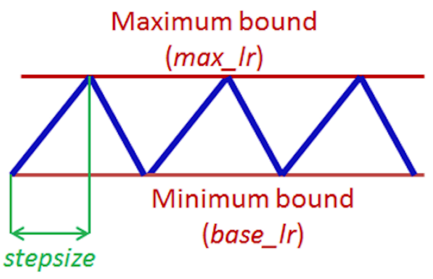


Figure C.1: Triangular learning rate policy, adopted from Smith [7]. Stepsize is the number of iterations in half a cycle.

The band of values between which to alternate is determined with the learning rate range test. A model is trained for a few epochs in which the learning rate is linearly increased from low and high values and plotted with respect to the training loss. The optimal learning rate range is where the loss starts to improve (and the curve starts to decrease) and before the loss starts diverging. The results of the learning rate range test for the different optimisers and batch size are shown in Figure C.2. The optimal learning rate ranges are therefore: [10**-7, 10**-4] for a batch size of 64 and [10**-5, 0.001] for a batch size of 256.

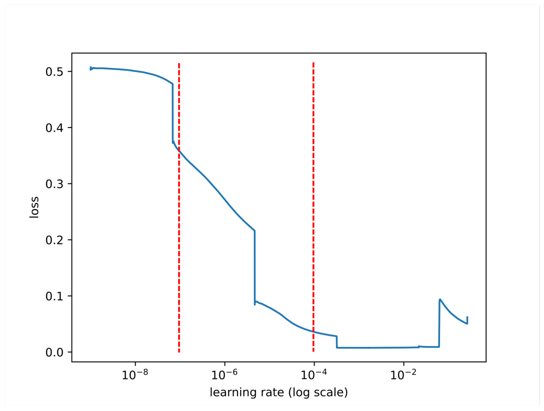
We both implement a cyclical learning rate without amplitude scaling (the values for the upper and lower bound remains the same) and with amplitude scaling (the values for the upper bound linearly decreased by half after each cycle). The final selection of values for the learning rate are shown in Table E.1.

C.3 Model parameters

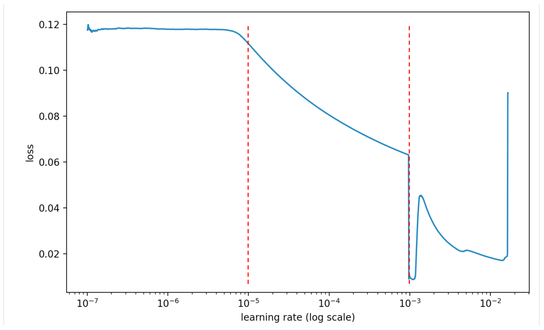
The model parameters that are discussed here are in with relation to the convolutional autoencoder, that is part of both model 1 and 2.

C.3.1 Convolutional layers

Complexity is directly increased when we add an extra layer. With the filters $[16, 32, 64, 128]$, padding 'same' and stride 2, the dense layer will be of size 8192 at 4 layers compared to $32,768$ at 3 layers (in Figure xx from the main text). The dense layer combines all the learned representation from the final convolutional layer from which the latent space can be learned. We will evaluate both cases and also the case with 5 layers, the dense layer will then be of size 2048.



(a)



(b)

Figure C.2: Learning rate range test for (a) a batch size of 64 and (b) a batch size of 256.

C.3.2 Kernel size

The kernels contain the weights that are learned to re-represent the data. Larger kernels are therefore more computational expensive. The size of the kernels describe the level of detail with with the input is scanned. Typical kernel sizes are 3×3 and 5×5, however to explore the effect of larger kernel sizes we will also adopt kernels with size 7×7.

C.3.3 Filters

Filters determine the number of representations that are being learned at each layer. The number of filters typically increase when the network goes deeper, and the intuition behind this is that at the beginning simple patterns are captured (e.g., edges, corners, etc.) and in subsequent layers those patterns are combined to make more complex patterns and the deeper we then go, the higher the amount of combinations that can be made. Therefore, we increase the number of filters with depth. When we add layers we start with a lower number of filters for the first layer.

C.3.4 Batch normalisation

In batch normalisation the features computed within a batch are normalised by the mean and variance. The input image is normalised and scaled between $[0, 1]$ and apart from that no normalisation or scaling takes place in subsequent layer. Deep learning models tend to work better with normalised data and to make sure that after each layer the data remains normalised, we apply batch normalisation. In batch normalisation the data is normalised within a batch.

C.3.5 Stride

The stride is a parameter that modifies the amount of movement over the image. For example, if stride is set to 1, than the filter will move one pixel at a time. The stride was set at 2, however, so the filters moves with two pixels at a time which also leads to smaller output image size at the next layer (with padding is zero the output image reduces by half in size).

C.3.6 Activation

An activation is the last component of the convolutional layer. This function adds a non-linear transformation to the output of the convolution. A typical chosen activation function is the rectified linear unit (ReLU). Next to ReLU we will also add leaky ReLU to the search space.

C.4 Clustering parameters

C.4.1 Number of clusters

For the two classification tasks in this study, the number of clusters was chosen to be between 5 and 7. Using three clusters for the first classification into three classes did not provide satisfactory performance, so a higher number of clusters was selected to give the model more flexibility. The elbow method also indicated that a higher

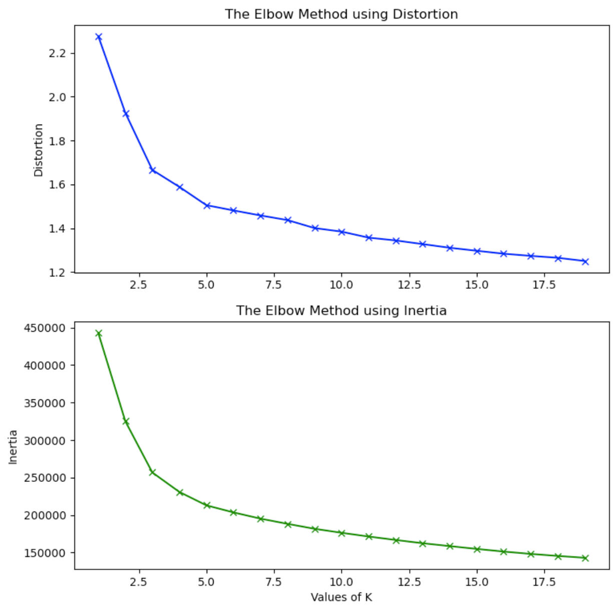


Figure C.3: Elbow plot

number of clusters would be more effective (as shown in Figure C.3). This method determines the optimal number of clusters by finding the point where distortion and inertia are as low as possible, but also significantly reduce from the previous point.

C.4.2 Gamma

The following range for gamma was selected: [0.05, 0.1, 0.2, 0.3, 0.4, 0.5]. In the paper from which we adapted this method, the value of gamma was set to 0.1 [1].

Bibliography

[1] Xifeng Guo, Xinwang Liu, En Zhu, and Jianping Yin. Deep Clustering with Convolutional Autoencoders. pages 373–382. 2017. doi: 10.1007/978-3-319-70096-0{_}39. URL `http://link.springer.com/10.1007/978-3-319-70096-0_39`.

[2] J Bergstra, D Yamins, and D D Cox. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. Technical report, 2013.

[3] D. Randall Wilson and Tony R. Martinez. The general inefficiency of batch training for gradient descent learning. *Neural Networks*, 16(10):1429–1451, 2003. ISSN 08936080. doi: 10.1016/S0893-6080(03)00138-2.

[4] Nitish Shirish Keskar, Jorge Nocedal, Ping Tak Peter Tang, Dheevatsa Mudigere, and Mikhail Smelyanskiy. On large-batch training for deep learning: Generalization gap and sharp minima. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, pages 1–16, 2017.

[5] Leslie N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay. pages 1–21, 2018. URL `http://arxiv.org/abs/1803.09820`.

[6] Jeremy Jordan. Setting the learning rate of your neural network., 3 2018. URL `https://www.jeremyjordan.me/nn-learning-rate/`.

[7] Leslie N. Smith. Cyclical learning rates for training neural networks. *Proceedings - 2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017*, (April):464–472, 2017. doi: 10.1109/WACV.2017.58.

[8] Hiroyuki Nodera, Yusuke Osaki, Hiroki Yamazaki, Atsuko Mori, Yuishin Izumi, and Ryuji Kaji. Classification of needle-EMG resting potentials by machine learning. *Muscle and Nerve*, 59(2):224–228, 2 2019. ISSN 10974598. doi: 10.1002/mus.26363.

[9] Hiroyuki Nodera, Yusuke Osaki, Hiroki Yamazaki, Atsuko Mori, Yuishin Izumi, and Ryuji Kaji. Deep learning for waveform identification of resting needle electromyography signals. *Clinical Neurophysiology*, 130(5):617–623, 5 2019. ISSN 18728952. doi: 10.1016/j.clinph.2019.01.024.

D Mel spectrograms

D.1 Mel spectrograms for Signal Type model

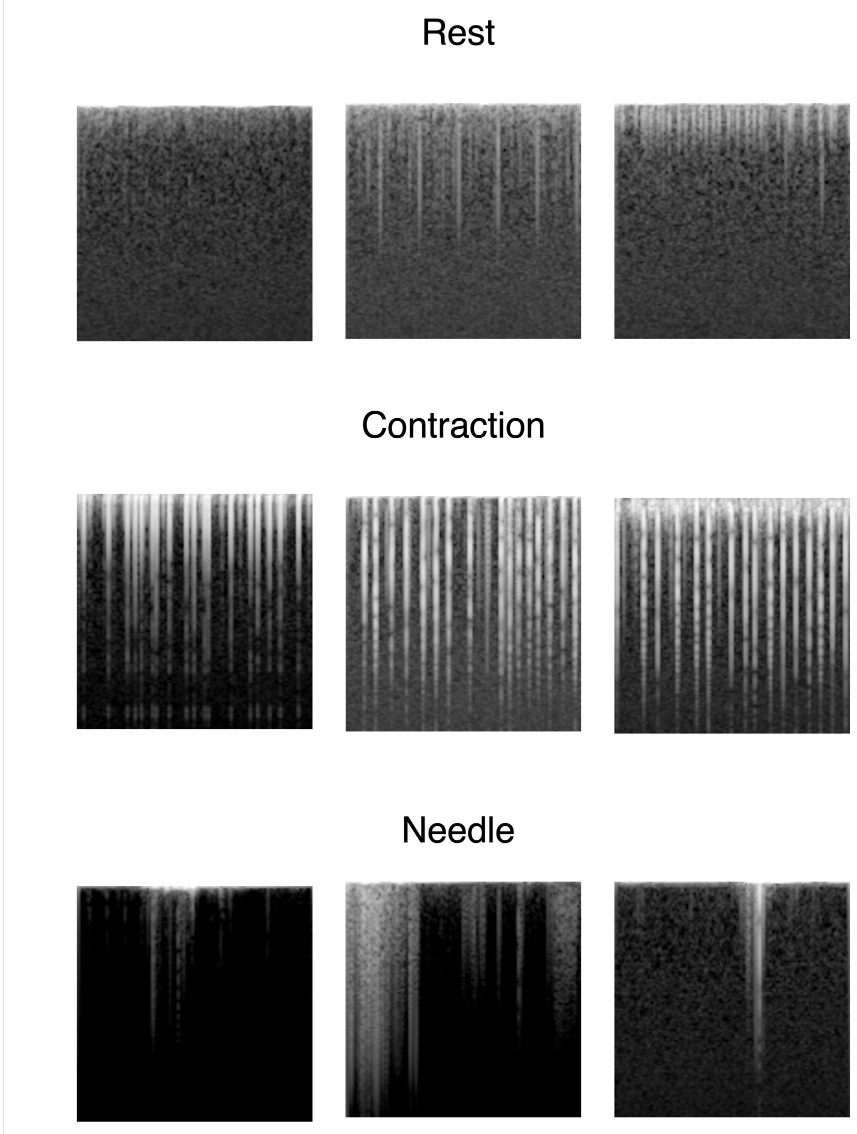


Figure D.1: Example of Mel spectrograms for the classes in the signal type labelled dataset: rest, contraction and needle. The Mel spectrogram image has time on the x-axis (max: 1.48 s) and frequencies on the y-axis with minimal frequencies at the top and maximum (10 kHz) frequencies at the bottom.

D.2 Mel spectrograms for Rest model

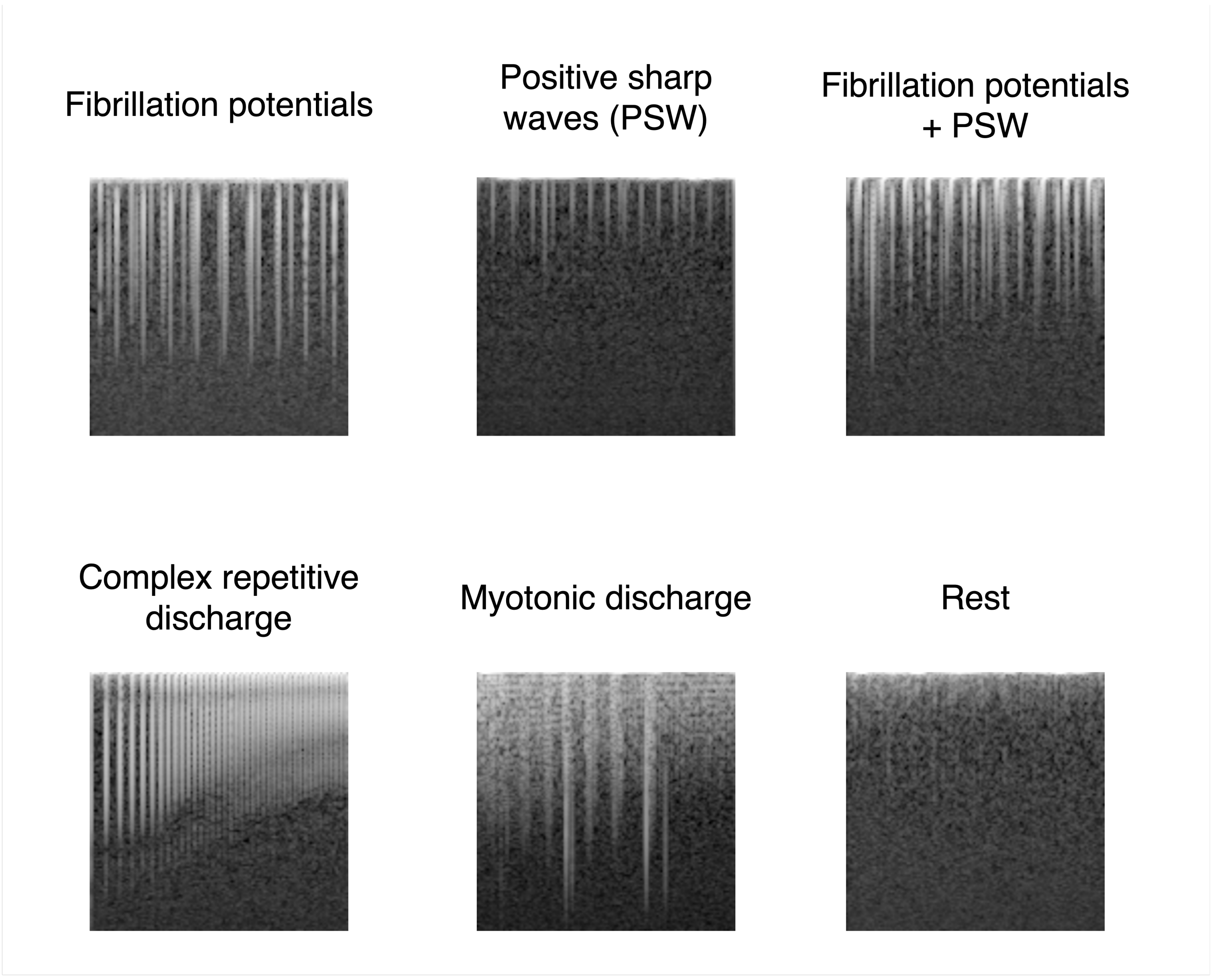


Figure D.2: Example of Mel spectrograms for the classes in the rest labelled dataset: fibrillation potentials, positive sharp waves, fibrillation potentials + positive sharp wave, complex repetitive discharge, myotonic discharge, and rest. The Mel spectrogram image has time on the x-axis (max: 1.48 s) and frequencies on the y-axis with minimal frequencies at the top and maximum (10 kHz) frequencies at the bottom.

D.3 Predicted Mel spectrograms for Signal Type models (Ia, IIa)

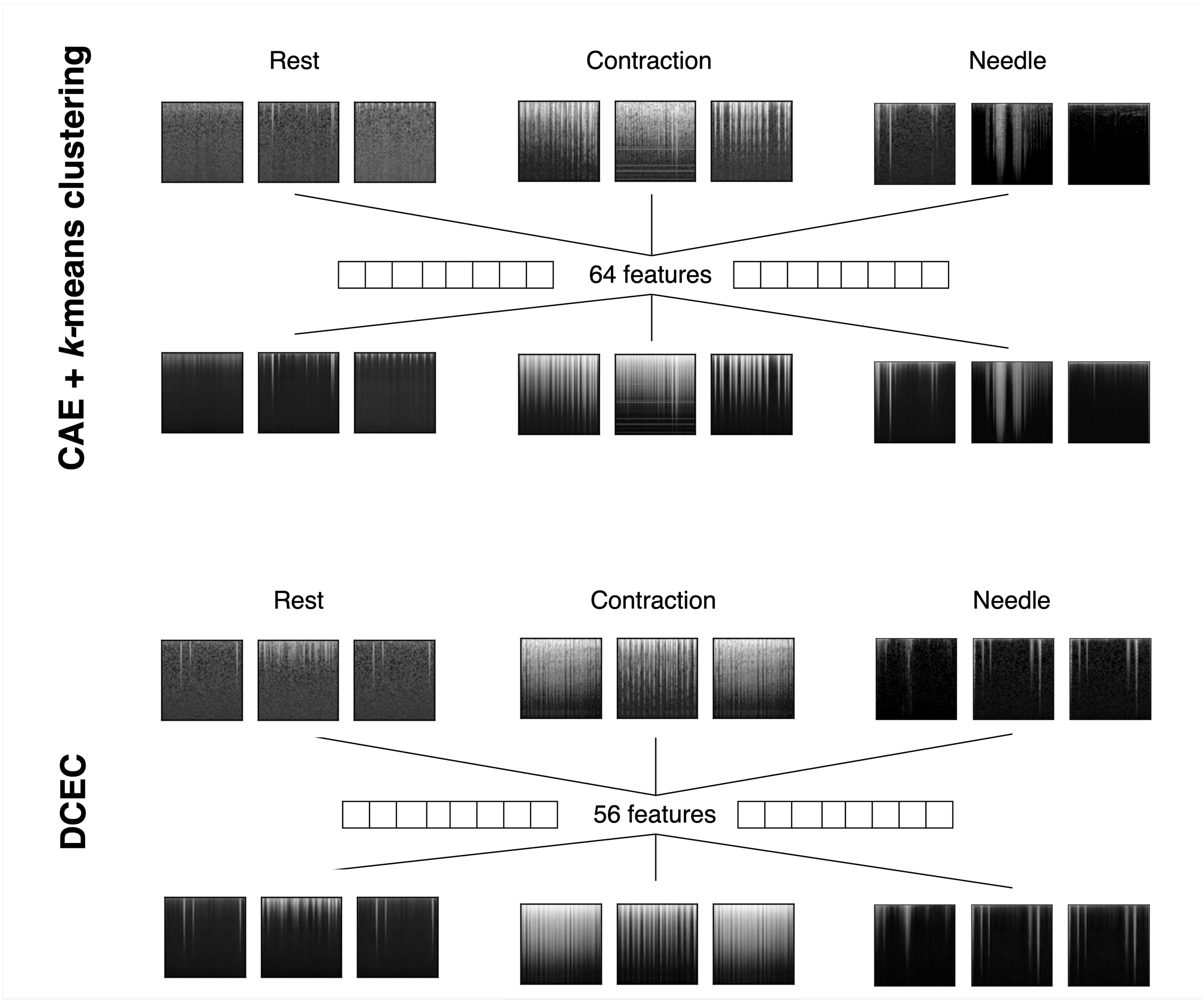


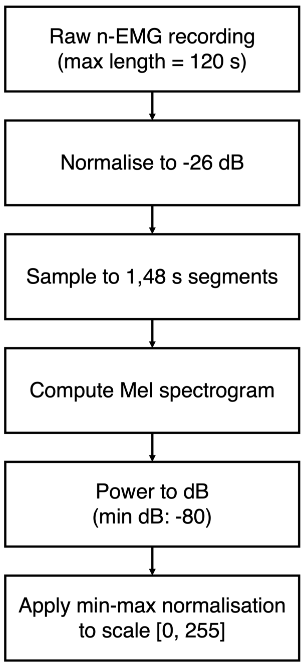
Figure D.3: Reconstructed outputs of model I (convolutional autoencoder (CAE) + *k*-means clustering) and model II (deep convolutional embedded clustering (DCEC). Examples are shown for each class. The middle part shows the number of features that are learned by each model to reconstruct the original image from.

D.4 Mel spectrograms: limitations of methods

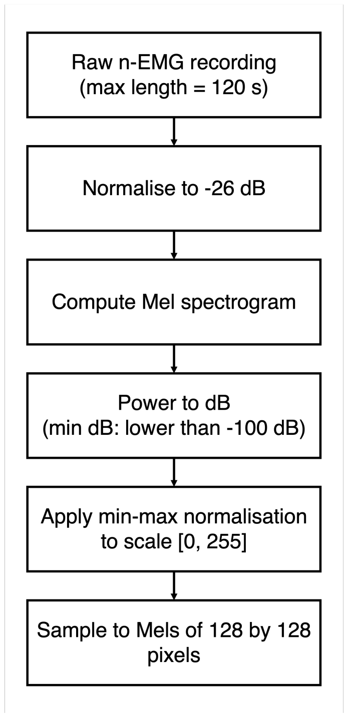
The methods for the creation of the Mel spectrograms was adopted from Nodera et al. [8, 9]. The exact methods are described in Section 4.1.4 of the main paper and the steps of this method are shown in Figure D.4a. However, this approach may lead to unwanted effects in the created Mel spectrograms. The pixel values in the Mel spectrograms represent the amplitude of a frequency on a specific time point. The pixel values in the current approach (approach a) are scaled to a range dependent on the amplitudes present in a 1.48 s segment. This effect is shown in Figure D.5: the top part of the figure shows a time signal of a needle EMG recording, the lower part of the figure shows Mel spectrograms that were created from 1.48 s time signal. The first row of Mel spectrograms were computed with approach a and it is seen that the pixel values in become darker for a large part of the signal when waveforms with higher amplitude come into view.

This effect may be unwanted, because it is desired that pixel values represent a similar amplitude. An alternative approach (approach b) would be to compute the Mel spectrogram over the full length of the signal (see order of steps in Figure D.4b) so that the pixel values remain the same and are independent of the 1.48 s time signal (Figure D.5).

Preliminary evaluations of the alternative approach did not perform better than the current approach in the classification of rest, contraction and needle Mel spectrograms.


Raw n-EMG recording (max length = 120 s)
Normalise to -26 dB
Sample to 1,48 s segments
Compute Mel spectrogram
Power to dB (min dB: -80)
Apply min-max normalisation to scale [0, 255]

(a)


Raw n-EMG recording (max length = 120 s)
Normalise to -26 dB
Compute Mel spectrogram
Power to dB (min dB: lower than -100 dB)
Apply min-max normalisation to scale [0, 255]
Sample to Mels of 128 by 128 pixels

(b)

Figure D.4: Order of steps for the computation of Mel spectrograms in the original approach (a) and a possible different approach (b).

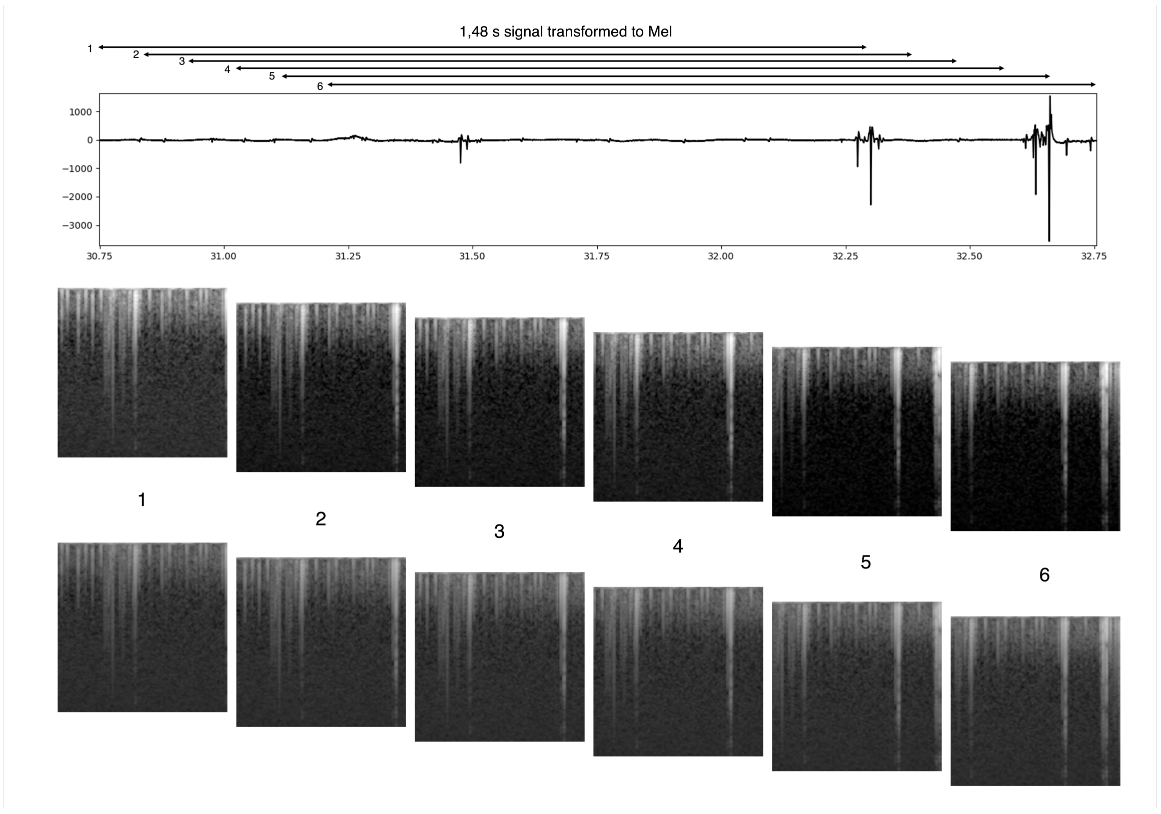


Figure D.5: Effect of two different approaches for the computation of Mel spectrograms. The upper Mel spectrograms correspond to approach (a) and the lower Mel spectrograms correspond to approach (b).

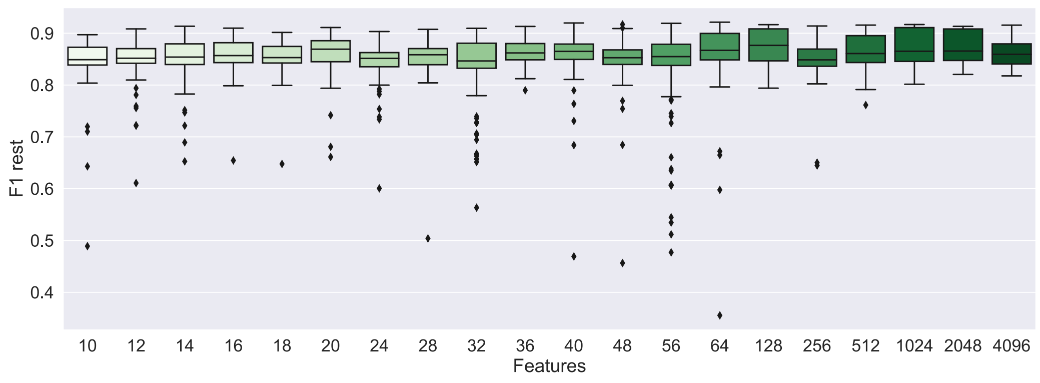
E Hyperparameter Optimisation: Results

E.1 Model 1: Convolutional Autoencoder + *k*-means clustering

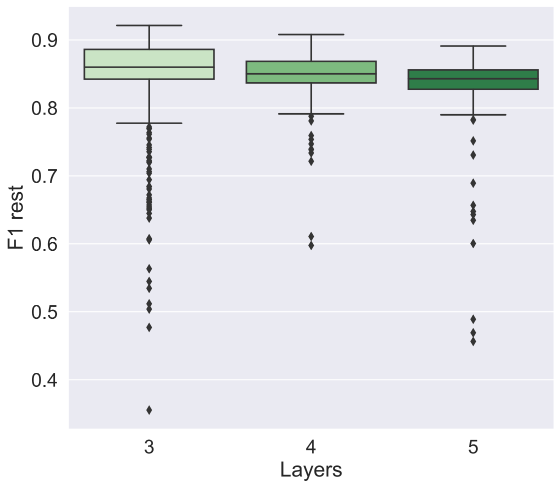
Table E.1: Best 20 results of hyperparameter optimisation for Convolutional Autoencoder (CAE). Each line shows a trial where the CAE is trained for 8 epochs and performance is determined with *k*-means clustering.

Performance		**Learning**				**Architecture**					**Clustering**	
F1 rest	Acc	Optim.	BS	Schedule	LR	Layers	Filters	Kernels	Activat.	BN	Feat.	*k*
0.921	0.930	adam	64	constant	0.0005	three	(32, 64, 128)	(7, 5, 3)	ReLU	True	64	6
0.920	0.925	adam	64	constant	0.0005	three	(32, 64, 128)	(5, 5, 3)	ReLU	True	40	6
0.919	0.926	adam	64	constant	0.0001	three	(32, 64, 128)	(5, 5, 3)	ReLU	True	56	6
0,919	0.926	adam	64	constant	0.0001	three	(32, 64, 128)	(5, 5, 3)	ReLU	True	56	6
0.918	0.923	adam	64	constant	0.0005	three	(32, 64, 128)	(5, 5, 3)	ReLU	True	64	6
0.918	0.923	adam	64	constant	0.0005	three	(32, 64, 128)	(7, 5, 3)	ReLU	True	64	6
0.917	0.919	adam	64	constant	0.0005	three	(32, 64, 128)	(5, 5, 3)	ReLU	True	64	6
0.917	0.928	adam	64	constant	0.0005	three	(32, 64, 128)	(5, 5, 3)	ReLU	True	56	6
0.917	0.922	adam	64	constant	0.0005	three	(32, 64, 128)	(5, 5, 3)	ReLU	True	48	6
0.917	0.925	adam	64	constant	0.0005	three	(32, 64, 128)	(7, 5, 3)	ReLU	True	40	6
0.917	0.923	adam	64	constant	0.0005	three	(32, 64, 128)	(5, 5, 3)	ReLU	True	1024	6
0.917	0.928	adam	64	constant	0.0005	three	(32, 64, 128)	(5, 5, 3)	ReLU	True	64	6
0.917	0.919	adam	64	constant	0.0005	three	(32, 64, 128)	(5, 5, 3)	ReLU	True	128	6
0.917	0.925	adam	64	constant	0.0005	three	(32, 64, 128)	(5, 5, 3)	ReLU	True	64	6
0.916	0.922	adam	64	constant	0.0005	three	(32, 64, 128)	(5, 5, 3)	ReLU	True	128	6
0.916	0,925	adam	64	constant	0.0005	three	(32, 64, 128)	(5, 5, 3)	ReLU	True	128	6
0.916	0.923	adam	64	constant	0.0005	three	(32, 64, 128)	(5, 5, 3)	ReLU	True	64	6
0.916	0.924	adam	64	constant	0.0005	three	(32, 64, 128)	(5, 5, 3)	ReLU	True	512	6
0.916	0.925	adam	64	constant	0.0005	three	(32, 64, 128)	(5, 5, 3)	ReLU	True	4096	6
0.915	0.926	adam	64	constant	0.0005	three	(32, 64, 128)	(7, 5, 3)	ReLU	True	64	6

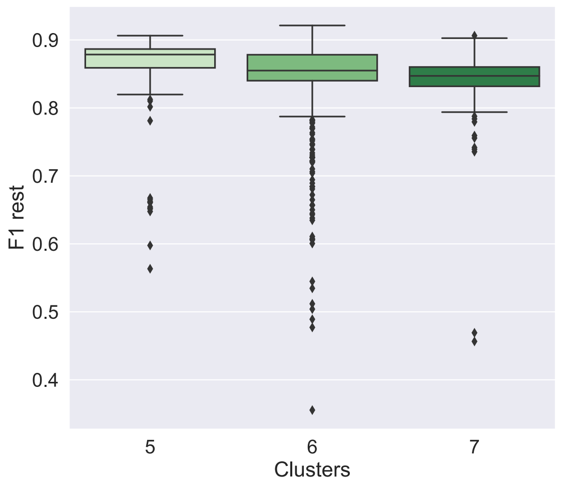
Acc. = accuracy; activat. = activation; BS = batch size; BN = batch normalisation; feat. = features; LR = learning rate; optim. = optimizer.



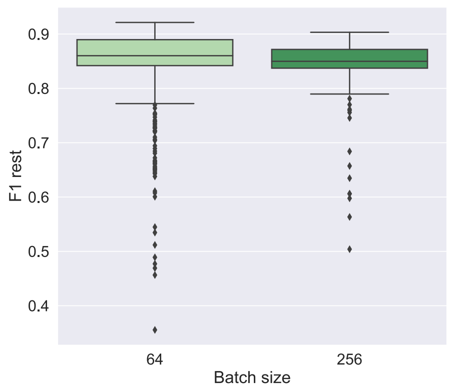
(a)



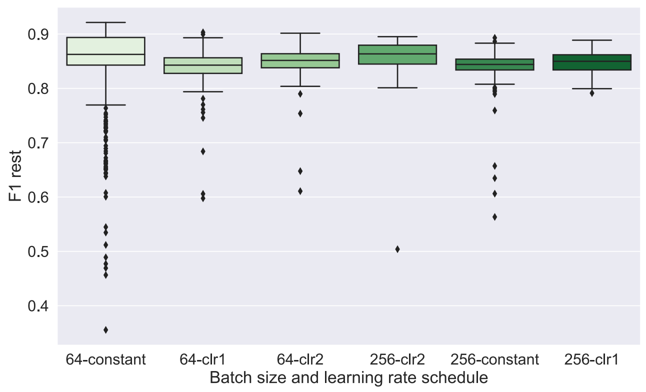
(b)



(c)



(d)



(e)

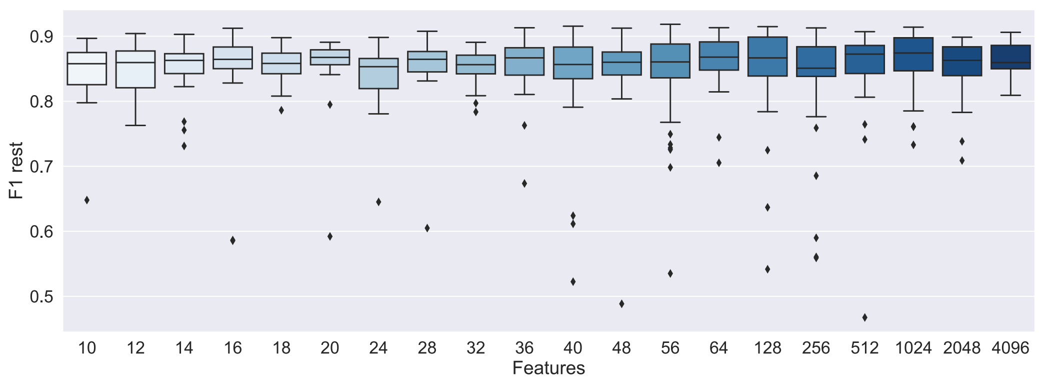
Figure E.1: Box plots with results for specific hyperparameters features (a), layers (b), clusters (c), batch size (d) or learning rate schedule (e) vs. performance (F1 rest). The box shows the quartiles of the data, the whiskers show the rest of the distribution except for outliers.

E.2 Model 2: Deep Convolutional Embedded Clustering

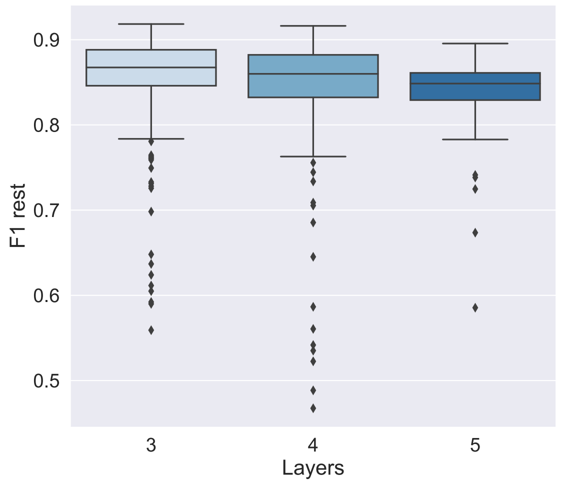
Table E.2: Best 20 results of hyperparameter optimisation for Deep Convolutional Embedded Clustering (DCEC). Each line shows a trial where the Convolutional Autoencoder is pre-trained for 8 epochs and the DCEC model is subsequently trained for 8 epochs.

Performance		**Learning**						**Architecture**						**Clustering**	
F1 rest	Acc	Optim.	BS	Schedule pre-train	LR	Schedule training	LR	Layers	Filters	Kernels	Activation	BN	Feat.	*k*	Gamma
0.918	0.925	adam	64	constant	0.001	clr1	(1e-07, 0.0001)	three	(32, 64, 128)	(7, 5, 3)	leakyReLU alpha=0.2	True	56	7	0,05
0.916	0.911	adam	64	constant	0.0001	clr1	(1e-07, 0.0001)	four	(32, 64, 128, 256)	(5, 5, 3, 3)	leakyReLU alpha=0.2	True	56	7	0,2
0.916	0.931	adam	64	constant	0.0001	clr1	(1e-07, 0.0001)	four	(16, 32, 64, 128)	(5, 5, 3, 3)	leakyReLU alpha=0.2	True	40	5	0,05
0.915	0.926	adam	64	constant	0.0001	clr1	(1e-07, 0.0001)	three	(32, 64, 128)	(7, 5, 3)	leakyReLU alpha=0.2	True	128	7	0,05
0.914	0.916	adam	64	constant	0.0001	clr1	(1e-07, 0.0001)	four	(32, 64, 128, 256)	(7, 5, 5, 3)	leakyReLU alpha=0.2	True	56	7	0,05
0.914	0.925	adam	64	constant	0.001	clr1	(1e-07, 0.0001)	three	(32, 64, 128)	(7, 5, 3)	leakyReLU alpha=0.2	True	56	7	0,05
0.914	0.920	adam	64	constant	0.001	clr1	(1e-07, 0.0001)	three	(32, 64, 128)	(7, 5, 3)	leakyReLU alpha=0.2	True	1024	7	0,2
0.913	0.921	adam	64	constant	0.0001	clr1	(1e-07, 0.0001)	three	(32, 64, 128)	(7, 5, 3)	leakyReLU alpha=0.2	True	64	7	0,05
0.913	0.922	adam	64	constant	0.001	clr1	(1e-07, 0.0001)	three	(32, 64, 128)	(7, 5, 3)	leakyReLU alpha=0.2	True	1024	7	0,2
0.913	0.901	adam	64	constant	0.0001	clr1	(1e-07, 0.0001)	four	(32, 64, 128, 256)	(7, 5, 5, 3)	leakyReLU alpha=0.2	True	36	7	0,05
0.913	0.917	adam	64	constant	0.0001	clr1	(1e-07, 0.0001)	three	(32, 64, 128)	(7, 5, 3)	leakyReLU alpha=0.2	True	64	7	0,05
0.913	0.920	adam	64	constant	0.0001	clr1	(1e-07, 0.0001)	four	(32, 64, 128, 256)	(5, 5, 3, 3)	leakyReLU alpha=0.2	True	128	7	0,1
0.913	0.916	adam	64	clr1	(1e-07, 0.0001)	constant	0.0001	three	(32, 64, 128)	(7, 5, 3)	leakyReLU alpha=0.2	True	256	7	0,2
0.913	0.920	adam	64	constant	0.001	clr1	(1e-07, 0.0001)	three	(32, 64, 128)	(7, 5, 3)	leakyReLU alpha=0.2	True	1024	7	0,2
0.912	0.920	adam	64	constant	0.0001	clr1	(1e-07, 0.0001)	four	(32, 64, 128, 256)	(7, 5, 5, 3)	leakyReLU alpha=0.2	True	48	7	0,05
0.912	0.920	adam	64	constant	0.0001	clr1	(1e-07, 0.0001)	three	(32, 64, 128)	(7, 5, 3)	leakyReLU alpha=0.1	True	16	7	0,2
0.912	0.922	adam	64	constant	5,00E-05	clr1	(1e-07, 0.0001)	three	(32, 64, 128)	(7, 5, 3)	leakyReLU alpha=0.2	True	56	7	0,2
0.912	0.921	adam	64	constant	0.001	clr1	(1e-07, 0.0001)	three	(32, 64, 128)	(7, 5, 3)	leakyReLU alpha=0.2	True	1024	7	0,2
0.912	0.924	adam	64	constant	0.0001	clr1	(1e-07, 0.0001)	four	(32, 64, 128, 256)	(5, 5, 3, 3)	leakyReLU alpha=0.1	True	56	7	0,4
0.912	0.911	adam	64	constant	0.001	clr1	(1e-07, 0.0001)	three	(32, 64, 128)	(7, 5, 3)	leakyReLU alpha=0.2	True	1024	7	0,2

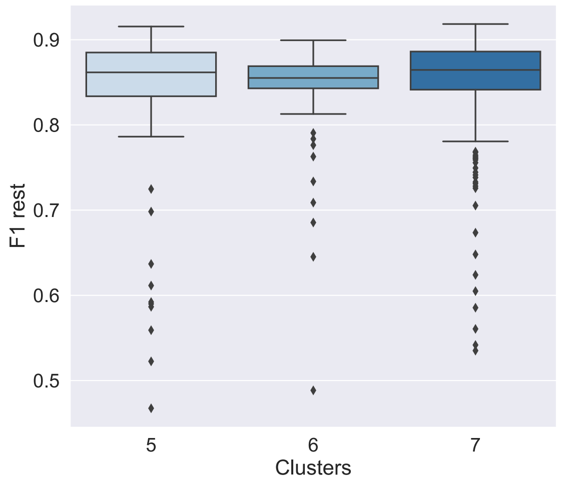
Acc. = accuracy; activat. = activation; BS = batch size; BN = batch normalisation; feat. = features; LR = learning rate; optim. = optimizer.



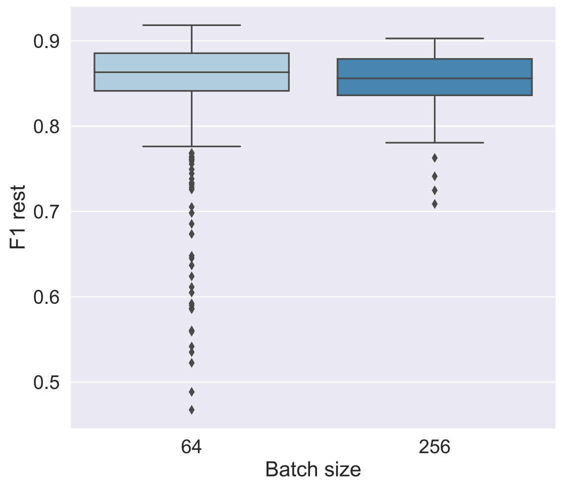
(a)



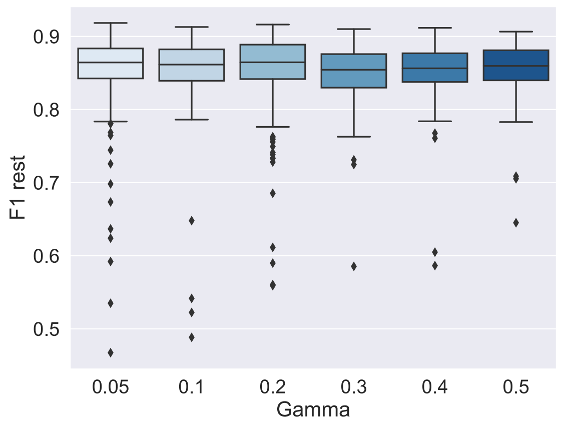
(b)



(c)



(d)



(e)

Figure E.2: Box plots with results for specific hyperparameters features (a), layers (b), clusters (c), batch size (d) or gamma (e) vs. performance (F1 rest). The box shows the quartiles of the data, the whiskers show the rest of the distribution except for outliers.

F Results interrater reliability per file

Table F.1: Results from the Krippendorff interrater reliability test, the values were computed in a previous project. Red numbers indicate poor interrater reliability, green numbers indicate excellent interrater variability and blue numbers indicate good interrater reliability (Deborah Hubers, Master thesis, april 2022).

File	Interrater variability
1	-0.09
2	0.98
3	0.99
4	0.70
5	0.74
6	0.09
7	0.95
8	0.86
9	0.13

G Additional Results

G.1 Training Deep Convolutional Embedded Clustering "best" parameters

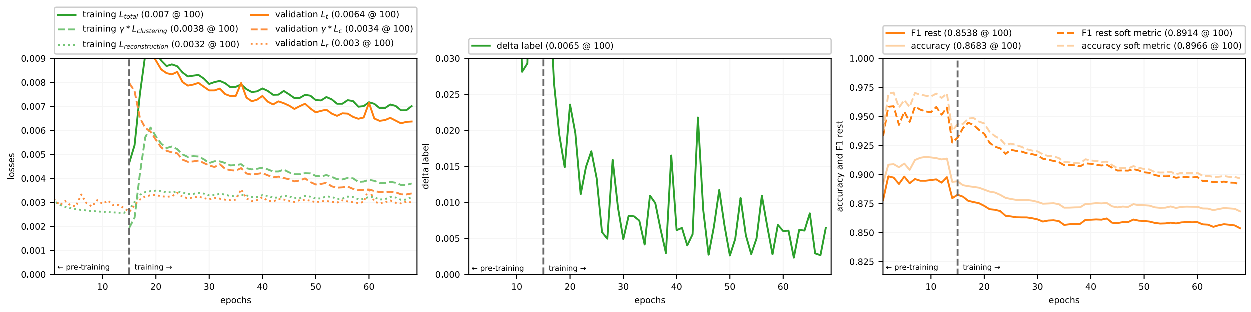


Figure G.1: Training results with all training data for deep convolutional embedded clustering for the classification of rest, contraction and needle signals. Based on these results it was decided to change the learning rate strategy from clr (without amplitude scaling) to clr2 (with amplitude scaling).

G.2 Classification of Labelled Rest Mel Spectrograms

Table G.1: Classification of labelled Mel spectrograms in Rest dataset as rest, contraction or needle movement by convolutional autoencoder + *k*-means clustering (model I). CRD = complex repetive discharge, PSW = positive sharp waves.

True label (*n* Mel spectrograms)	Rest (soft metric) [%]	Contraction (soft metric) [%]	Needle movement (soft metric) [%]
Rest (767)	72 (69)	27 (14)	0 (0)
Fibrillation potentials (216)	63 (20)	37 (16)	0 (0)
PSW (388)	90 (86)	10 (4)	1 (1)
Fibrillation potentials and PSW (163)	31 (25)	69 (1)	0 (0)
CRD (272)	67 (26)	33 (13)	0 (0)
Myotonic discharge (9)	0 (0)	78 (30)	22 (11)
Total (1815)	70 (56)	29 (11)	0 (0)

Table G.2: Classification of labelled Mel spectrograms in Rest dataset as rest, contraction or needle movement by deep convolutional embedded clustering (model II). CRD = complex repetive discharge, PSW = positive sharp waves.

True label (*n* Mel spectrograms)	Rest (soft metric) [%]	Contraction (soft metric) [%]	Needle movement (soft metric) [%]
Rest (767)	80 (65)	20 (13)	0 (0)
Fibrillation potentials (216)	53 (10)	46 (34)	0 (0)
PSW (388)	92 (87)	5 (1)	3 (1)
Fibrillation potentials and PSW (163)	52 (12)	48 (1)	0 (0)
CRD (272)	42 (22)	46 (14)	12 (3)
Myotonic discharge (9)	0 (0)	44 (22)	56 (44)
Total (1815)	70 (52)	26 (12)	3 (1)

G.3 F1 During Training Rest Models

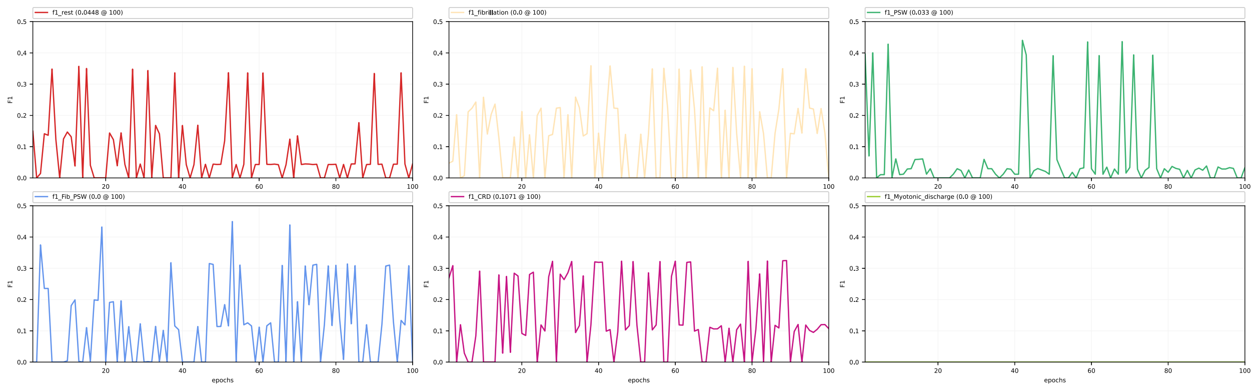


Figure G.2: F1 scores of all classes during training of convolutional autoencoder + k-means clustering. F1 score for myotonic discharge class was not available during training because these Mel spectrograms were not classified as rest.

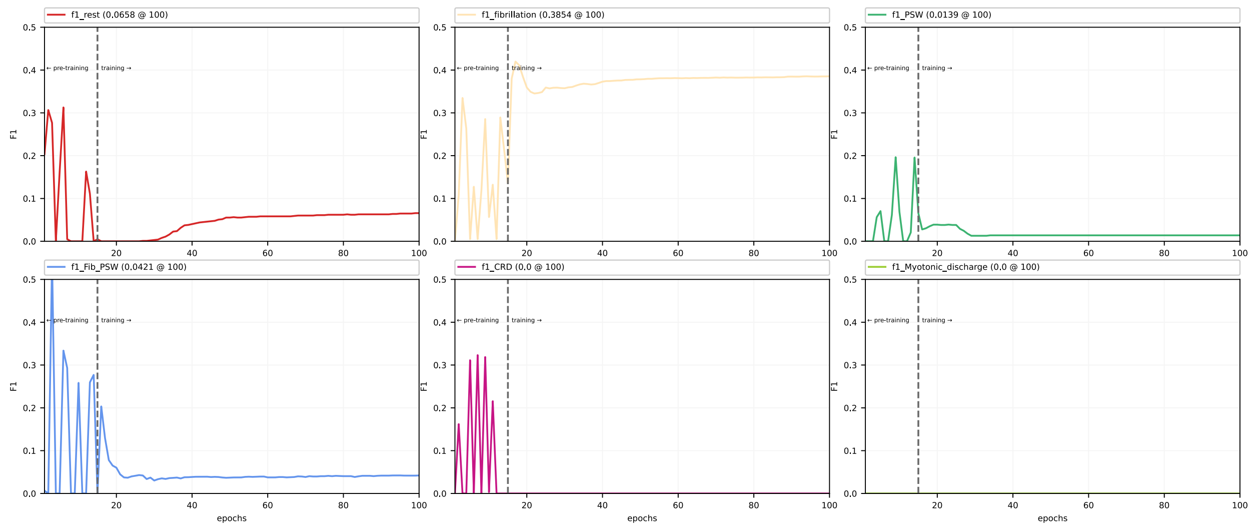


Figure G.3: F1 scores of all classes during training of deep convolutional embedded clustering (model II). F1 score for myotonic discharge class was not available during training because these Mel spectrograms were not classified as rest.