Towards speech-based brain-computer interfaces: finding most distinguishable word articulations with autoencoders

Eli Stolwijk
6000738
Master Artificial Intelligence
Faculty of Science
Utrecht University
Netherlands

Julia Berezutskaya
University Medial Centre Utrecht
Daily Supervisor

Chris Klink
University Utrecht
First Supervisor

Ben Harvey
University Utrecht
Second Supervisor

Universiteit Utrecht

December 23, 2022

1 Abstract

People that suffer from locked-in syndrome are severely limited in their means of communication. Recent advances in BCI technology have allowed communication by typing letters on a screen through the decoding of brain activity. However, direct word decoding, meaning decoding whole words at a time instead of characters, can provide a big increase in communication speed and efficiency. To find out which words are most suitable for such applications we want to find the words for which the neural activation of their attempted articulation are most easily distinguishable. Instead of measuring differences in brain activity directly, we will measure the differences in the patterns of muscle movements during articulation and use those as representatives for the brain activity instead. We will find the set of most distinct words by using two neural network autoencoder architectures to condense rtMRI videos of speech into representative vectors. We then cluster these vectors into 20 clusters and extract the representatives of each cluster to get the set of 20 most distinct words in their articulation. We will try two different autoencoder architectures, one using only 3-dimensional convolutions (3D-CNN) and the other using a combination of 3-dimensional convolutions and GRU cells (ConvGRU). To nudge the models into learning relevant word embeddings, we introduced phonemic information of the word labels in two different ways. The first being a one hot encoded phoneme content vector for each word, and the second being a custom component to the loss function that incorporated a phonemic distance metric inspired by the classic Levenshtein distance. We found that both architectures were capable of generating relevant word embeddings while reconstructing the input of rtMRI videos of speech. We further found that the ConvGRU outperformed the 3D-CNN on almost every metric discussed. Additionally, we found that the results from the ConvGRU generalized well over multiple participants suggesting a generalizability to people with LIS. The set of 20 words generated by the ConvGRU made sense on informal and formal inspection of the cluster space and were therefore chosen as the set of most distinct words for future direct word decoding BCI applications.

Contents

2 Introduction

Communication is an essential part of being human. People that become paralyzed can completely lose the ability to communicate with the outside world, they become locked-in. The term locked-in syndrome (LIS) describes people that are fully conscious but have no means of voluntary muscle control, preventing them from producing speech, limb or facial movements (Lulé et al., 2009). LIS can be caused by many different conditions. The most frequent causes of LIS are vascular-related with the most frequent cause being brain stem stroke (Vidal, 2020). LIS is also observed in the late stages of neurodegenerative diseases like amytrophic lateral sclerosis (ALS). Other more rare etiologies include drug abuse, head trauma, tumors, encephalitis, arthritis and toxin exposure (Patterson and Grabois, 1986).

The LIS can be divided into three categories: Classical, Incomplete and Total (Bauer et al., 1979). Classic LIS is characterized by quadriplegia (paralysis of all four limbs and torso) and aphonia (inability to produce sound) with preserved consciousness, vertical eye-movements and blinking. Incomplete LIS is characterized by the same characteristics as Classic LIS, but with the addition of some preservation of voluntary movement other than vertical eye movements. Total LIS is characterized by a complete immobility, including the eyes.

Unlike common belief, people with LIS can still live a happy life. Multiple studies have shown that people with LIS report a similar quality of life (QoL) compared to age-matched healthy individuals (Rabkin et al. (2000); Kübler et al. (2005); Laureys et al. (2005)). Furthermore, the 10-year survival rate of people with LIS is over 80% (Doble et al., 2003)), with some returning back to work (Smith and Delargy, 2005), and some even writing a book (Bauby, 2008). Thus, despite their severe handicap, LIS patients can still live a life worth living. Albrecht and Devlieger (1999) found that the main determinant for the QoL of people with severe paralysis is the subjective feeling of control over their life. In order to obtain this feeling, it is essential for locked-in people to be able to communicate with their surroundings. Furthermore, Rousseau et al. (2015) found that sociodemographic variables such as gender and education level, which traditionally influence QoL, were not found as factors of the QoL in people with LIS. Instead, they found that the restriction on their communication had the most significant (negative) association with QoL. This is further backed up by Bruno et al. (2011), who found that the ability to produce speech is among the main predictors for happiness in people with LIS.

Classically, the most used form of communication with locked-in people is through some sort of code using the eyelids. It may be clear that this method is not an option for people with Total LIS and even for those locked-in that can use this blinking code, communication is slow and inefficient. Usually the blinking code has to be initiated by a care giver and is limited to answering only binary questions (e.g. blink once for yes and twice for no) (Olivia Gosseries et al., 2009). More expressive ways of communication using only the eyelids are possible, but the increase in expressivity usually comes with a decrease in communication speed. An often used method involves selecting one letter at a

time by blinking to indicate when a caregiver should stop scrolling through a set of letters (León-Carrión et al., 2002). Though infinitely expressive, this method is very slow. As the restriction on communication has one of the most significant negative affects on QoL, one way to improve the QoL for people with LIS is to enhance their communication capabilities through the means of electronic aid devices. Such devices can provide quicker and more expressive ways of communication. Additionally, electronic aid devices can allow someone with LIS to initiate communication independently, where traditional communication through blinking requires a caregiver to pay attention. Such electronic devices can incorporate eye trackers (Yumang et al., 2020) or exploit possible remnants of voluntary movement, for example by using a mouthstick (Smith and Delargy, 2005). However, in the last few decades technology has allowed applications that locked-in patients can control with their brain directly. This paper will focus on such devices, which interface directly with the brain, also know as Brain Computer Interfaces (BCI).

Since LIS does not necessarily involve degradation of grey matter itself, neural patterns of attempted movement can still be observed in the motor cortex. Locked-in people with vascular causes usually suffer from damage in the pathways leading out from the brain, not the brain itself. In the case of degenerative diseases, like ALS, it is not entirely clear how the motor degradation occurs, but Pandarinath et al. (2015) showed that motor cortex signals in ALS patients were comparable to healthy non-human primates suggesting that, despite the neural degradation, the motor cortex may be able to retain its core functionality. Therefore, when someone who is locked in attempts to move a particular set of muscles (for example raise their arm), it is expected that neural activation in the brain occurs in a similar way as in a healthy individual. Based on this assumption, the brain activity of attempted movement should be able to be detected and decoded into physical behaviour by a BCI.

There are many different kinds of BCIs, however for LIS patients, the most important one is the communication BCI (Wolpaw, 2007). A communication BCI is a device that attempts to decode neural activity of the brain into physical behaviour. An example of this would be a program that cycles through a set of letters and a BCI user selecting the currently shown letter by attempting some kind of movement, e.g. raising an arm. The BCI recognizes this neural activity, selects the current letter and continues scrolling until further input by the user. In the last few decades, many communication BCIs have been proposed, but with mixed success. Depending on the measurement techniques, hardware of the system, software of the system and the abilities of the user, many trade-offs have to be made regarding performance, reliability, sustainability and invasiveness.

Previous communication BCIs have mostly worked by enabling on-screen typing or writing, using individual characters (Gilja et al., 2015; Vansteensel et al., 2016; Nuyujukian et al., 2018). Recently some advances have been made using a different type of character based BCI that uses attempted handwriting (Willett et al., 2021) (90 characters per minute with 94 percent accuracy). However, a BCI that works by decoding entire words at a time can provide a faster and more natural way of communication. Moses et al. (2021) have

shown promising results on direct word decoding, achieving 15 decoded words per minute with 75 percent accuracy.

Although these results are promising, there is possibility for an efficiency increase by using only sets of words that are theoretically most dissimilar in the neural activity of their attempted articulation. More distinct neural patterns mean higher distinguishability and therefore provide a more reliable decoding. Previous work has shown that different neural patterns in the motor cortex correspond to different motor patterns in the muscles, including facial muscles, during speech production (Bouchard et al., 2013; Chartier et al., 2018; Mugler et al., 2018). This research suggests that neural patterns in the motor cortex reflect the spatial organization of body parts and associated muscles. In other words, each unique pattern of muscle movement arises from a unique pattern of activation in the brain. This implies that there is a unique pair of neural activation and muscle movements for each word in speech. This allows us to use the pattern of muscle movements (articulation pattern) as a representative for the pattern of neural activation (neural pattern). Therefore, following this assumption, identifying a closed set of words with most distinct patterns in the muscle movements of their articulation by healthy individuals provides us the set of words that are most distinct in the neural pattern of their attempted speech, and thus the set of words that should be most reliably decoded by a BCI application.

Achieving a representation of an articulation pattern requires information about the movements of all the muscles that constitute it, at each time step. Moreover, articulation patterns cant be measured directly on people that are locked-in due to their inability to produce speech, so the resulting set of distinct articulation patterns found on healthy individuals should generalize well across multiple people. When results are specific to each participant separately, their is no point in using the findings of a healthy participant on someone who is locked-in. When results generalize over multiple healthy people we can be more confident that these results will also generalize over to people with LIS.

There are multiple measurement techniques that can capture the movements of (parts of) the mouth when producing speech. Ultrasound probes provide a non-invasive technique to visualize muscles during articulation, but they are limited to only a small area, usually just the contours of the tongue (Akgul et al., 1998; Wilson, 2014; Saito et al., 2021). Electromyography (EMG) measures electric muscle activity through sensors that are placed on the skin. This technique can cover a wider area of the face than ultrasound, but misses the muscles within the mouth and throat (Honda, 1983; Schultz and Wand, 2010). Electromagnetic Articulography (EMA) uses an electromagnetic field to capture the movements of electrodes within and outside the mouth (Schönle et al., 1987; Rebernik et al., 2021). Since the electrodes can be placed on the muscles within the mouth and on the skin of the throat, EMA is able to follow most of the muscles that form the articulation pattern. However, as there is only a (small) fixed amount of electrodes, the coverage is limited to the points where an electrode is attached, all other information is lost. Moreover, the placement of electrodes on inner mouth surfaces like the tongue is likely to alter the articulation pattern

(Katz et al., 2006).

Magnetic Resonance Imaging (MRI) is a technique that measures how different tissues react to a strong magnetic field, thereby creating clear pictures of the body parts within the scanner (Hoult and Bhakar, 1997). Recent advances in MRI technology have allowed the capture of processes in real time. The advantage of real-time MRI (rtMRI) for the extraction of articulation patterns is that the footage of the mid-sagittal slice contains a clear view of all the muscles used during articulation at many frames per second. Hence, a minimal amount of information is lost during measurement (Csapó, 2020). Moreover, the non-invasive nature and the high capture quality make mid-sagittal rtMRI footage the most suitable measurement technique for the extraction of articulation patterns.

The most straightforward way to identify which articulation patterns are similar and thus which ones are dissimilar is by clustering them. However, mid sagittal rtMRI video data is complex and high-dimensional, describing complex spatio-temporal dynamics. Therefore, raw rtMRI data may not be suitable for conventional clustering (Assent, 2012). Therefore, clustering the articulation patterns can only be done effectively when the rtMRI videos are reduced in dimensionality. Many dimensionality reduction methods exist, however autoencoders have shown the most promise extracting meaningful features in image processing (Meng et al., 2018). An autoencoder consists of an encoder and a decoder, usually made up of neural networks. The idea of an autoencoder is that you give it an input, let the encoder learn a condensed representation of that input and then let the decoder reconstruct the original input only from that representation. After reconstruction, the difference between the input and the reconstructed input is used to adjust the parameters of the model. The layer between the encoder and decoder is called the bottleneck layer. The bottleneck layer is where the input has been most condensed. Since this condensed representation contains all information necessary for the reconstruction of the input, it contains all the representative features of the input, despite being reduced in dimensionality. Hence, by extracting the bottleneck layer representation of the input, an autoencoder can be used as a dimensionality reduction method (Bank et al., 2020).

Since rtMRI videos of mid sagittal slices constitute 3-dimensional data (2 spatial, 1 temporal and no color channels), the encoder and decoder need to be able to capture dependencies across all three dimensions. Recent advances in the field of computer vision have allowed for neural networks to be more adept at handling three dimensional input. The two main methods of dealing with three dimensional input is to either use only 3-dimensional convolutional operations (Ji et al., 2013) or to use some combination of convolutional operations and recurrent neural networks (Vinyals et al., 2014; Shi et al., 2015). Many studies have already successfully incorporated three-dimensional neural networks within autoencoder frameworks in many different domains (Srivastava et al., 2015; Haugen et al., 2019; Dastider et al., 2021). This suggests that such architectures generalise well across domains providing the motivation for us to use these architectures for our approach.

The present study aims to identify a set of 20 words that are most

distinct in terms of their articulation pattern, extracted from mid-sagittal rtMRI footage. A set of 20 words will allow 20 degrees of freedom in tasks such as selecting menu items or giving commands, while remaining a manageable number for the development and implementation of a speech BCI. We are not aiming for the incorporation of full languages yet, as this would lead to highly complex and unreliable applications. Instead, we are aiming at an efficiency increase for direct word decoding BCI applications that incorporate small sets of words to provide the user with a more limited but more reliable application. Narayanan et al. (2014) has provided a dataset (USC-TIMIT) of mid-sagittal rtMRI footage of 10 participants during a speech production task, in which they read sentences from the TIMIT dataset. In order to cluster the words effectively, we will reduce the rtMRI videos in dimensionality using two different autoencoder architectures. One architecture uses only 3-dimensional convolutions (3D-CNN) and the other uses a combination of 3-dimensional convolutions and recurrent neural networks (ConvGRU). After each word is reduced to a representative vector, we cluster the vectors into 20 clusters and extract the representatives of each cluster and present them as the 20 most distinct words in their neural/articulation patterns.

3 Method

3.1 Preprocessing

Figure 1: Mid sigattal slice extracted from a video of participant F1

For our research we used the freely available USC-TIMIT dataset, provided by Narayanan et al. (2014). This dataset consists of both real time Resonance Magnetic Imaging (rtMRI) and ElectroMagnetic Articulography (EMA) data collected in healthy human subjects who read English sentences out loud. We will use the rtMRI data as input and output for our models and the EMA data as a correlation metric to evaluate model performance. The MRI data consists of rtMRI footage of the mid-sagittal slices (see Figure 1) of 10 participants (5 male, 5 female), all speaking the same set of 460 sentences. For each participant, the footage is separated into videos of 5 sentences, about 20 to 30 seconds long with

a frame rate of 23.18 frames per second and a resolution of 68 x 68 pixels. Audio was recorded simultaneously with the MRI recording and was later denoised and synchronized to the footage. The dataset also provides a transcription of the produced speech at the level of sentences, words and phonemes.

The original videos in the dataset each contain 5 full sentences. First, we split these videos to contain only one word each. This was done by selecting the frame that corresponds to the beginning of a word and the frame that corresponds to the end of a word according to the transcription. All frames in between these two frames are extracted and saved with the label of the respective word. It is worth noting that there are duplicate words between sentences, so there are instances where multiple videos have the same label. These instances can be used during the evaluation of the model performance as identical words should have high similarity in their embedding space. Furthermore, because of the inter-subject differences in facial anatomy and positioning in the scanner we train separate models for each participant, similar to Csapó (2020) and Yu et al. (2021).

3.1.1 Feature reduction

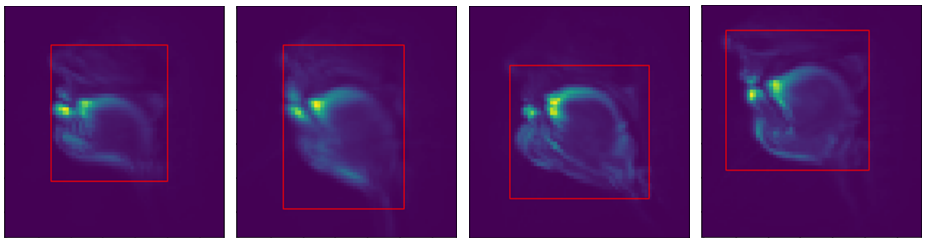


Figure 2: Variance heat maps for participants F1, F2, M2, M3 respectively. The red rectangles are the borders that contain all pixels with above average variance

A large portion of the pixels in the videos represent empty space (see Figure 1). These pixels don't contribute any information about the articulation patterns. As every pixel is a feature to the model, we don't want to include all these non-informative features. To exclude the non-informative features we need to determine which pixels are informative and which are not. First, we calculate the variance of each pixel for every video, effectively creating a variance heat map per video. Then we sum all the pixel variance values, for every variance heat map, creating a variance heat map over all videos for each participant. Then, we calculate the average value for each pixel. Then we calculate the minimal frame size (height x width) that still contains all above-average variance pixels for each participant. Therefore, we calculate the 4 borders (upper, bottom, left, right) that are as far away from the original 68 x 68 border, that still include all pixels with above average variance. Figure 2 shows some examples of these minimum frame sizes for some of the participants. Since we want to use

the same frame size for all participants to allow for transfer learning, we take the largest minimal height (participant F2) and largest minimal width (participant M3) and come to a universally required minimum frame size of 48 x 44. This reduces the number of input features per frame of a video from 4624 to 2112, effectively halving the amount of input features without the loss of any above-average informative feature.

Originally the videos are in the RGB color scheme. However, as the videos are in black and white, this extra color channel dimension provides no useful information. Therefore, we will convert the RGB videos to gray scale, further reducing the feature space by 66 percent (3 color channels becoming 1) without any decrease in representational power.

3.1.2 Padding

One important property of the 3D-CNN architecture that we are going to use, is the requirement of a fixed input size. The input size being fixed is not a problem for the height and width dimensions of our data as these are fixed to 48 x 44. The time dimension however will vary for each word as different words take different amounts of time to be articulated. This results in our data having different amounts of frames per video. To make the data suitable for a 3D-CNN architecture we will need to use padding. Padding consists of adding non-expressive data around the shorter sequences to in essence 'fill up' the sequence until it is the required size. In our case, this means that we will have to add frames containing all zeros to the original video until it has the same amount of frames as the video with the highest amount of frames in our data. For example, assuming the longest video in our data is 10 frames long, we have to add all-zeros frames to all videos that do not have 10 original frames, until the original and padded frames of each video combine to a count of 10. We add padded frames to the left and right equally meaning that the original frames will be in the centre of the padded sequence. I.e., in our previous example a video of 4 original frames will have 3 padded frames to the left and 3 padded frames to the right.

However, padding can come with a cost (Dwarampudi and Reddy, 2019; Lopez-del Rio et al., 2020). Since padding adds data (0's), it slightly alters the original input. This does not necessarily have to come with a decrease in performance, however as the amount of padding increases it is more likely for a decrease in performance to be observed. For example, a video of 99 original frames and 1 padded frame will be expected to suffer a low padding costs as only 1 percent of the data is non-expressive. A video of 10 original frames and 90 padded frames however is expected to have a relatively high padding costs as the data consists for 90 percent of 0's. To circumvent the problem of padding, we used a second architecture that uses Recurrent Neural Networks (RNN). RNN's do not suffer from the necessity of padding since nodes in an RNN are allowed to create connections with themselves. Due to these cycles, the network can feed into a next time step instead of a next layer, thereby allowing for the processing of sequences of arbitrary length.

3.1.3 Frame counts

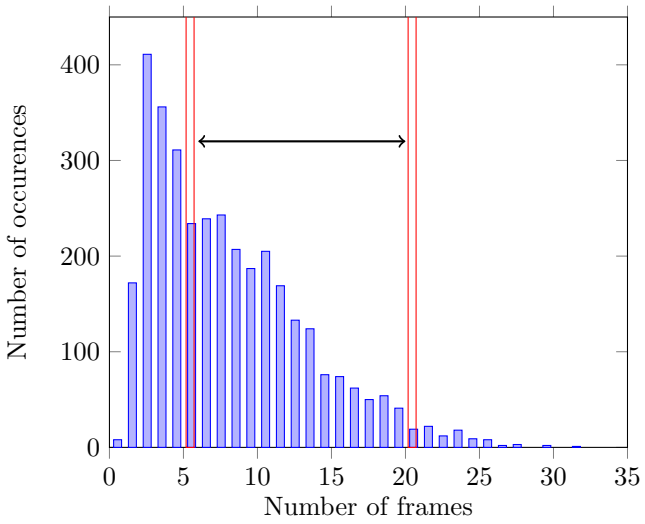


Figure 3: Frame count histogram for participant F1. Words outside the red lines were excluded from this study

The pronunciation of different words takes different amounts of time based on the length of the word, context within the sentence and reading speed of the speaker. We need to determine to what size we fix the time dimension. If we choose the size of the time dimension to be too large, the smaller words will need to have too much padding. If we make the time dimension too small, we might have to exclude too much of our data to successfully train our model. Figure 3 shows the distribution of frames for participant F1. We can see that videos with low frame counts occur the most. However, due to their low frame count they don't contain much information. Recall that the videos were shot at 23.18 frames per second resulting in a little over 0.04 seconds per frame. Videos with 1 frame are not videos but images so we will exclude those. Videos with 2, 3 and 4 frames contain so few frames, and thus such little information, that it is not worth including them into our data as the padded frames will dominate the original sequence. Therefore, we only considered videos with at least 5 frames. We think this boundary is high enough to get meaningful information within the videos and low enough to include enough data points to successfully train our model. For the upper boundary, we decided t draw the line at 20 frames. From this point, the benefit of additional data points does not outweigh the cost of the required additional padding on all other words. Thus, we decided to include all videos with $5 \leq$ frame counts ≤ 20, resulting in a dataset of 1904 videos (out of an original 3453). For all other participants the distributions followed similar patterns as shown in Figure 3, so we used these frame count cutoffs for all participants.

3.1.4 Phonemes

Neural networks can learn in many different ways. However, for this research, we want our model to behave in a way that is useful for our goal of finding distinct articulation patterns. In other words, we want the model to learn embeddings that represent specific articulation properties from their corresponding words. Therefore, we implemented a second data stream to nudge the model in this direction. We did this by adding a one-hot encoding of the phonemic content of the written word as additional input for the last linear layer. Furthermore, we also incorporated the phonemic content of the word into a custom loss function, again, to try to nudge the model into learning relevant features for our research. To get the phonemic content of each word, we used the open-source nltk cmudict library (Wagner, 2010). Consequently, every word that was not present in this library had to be excluded from the dataset, leaving us with 1885 videos for participant F1. Some of the other participants were left with more words (max = 2225), some with much less due to errors in the data acquisition (min = 1219), however this did not impact the current study as we only needed small sets (< 650) of words for all participants other than F1 (see Section 2.5).

Much like the variability in video length, there is also a variability in word length, and more specifically, phoneme count. In our second data stream we feed the model extra information about the phoneme content. We do this by one hot encoding the phonemes present in the label. This means that each phoneme has an index and whenever that phoneme is present, the value of that index is 1 and all others are 0. Given that we have 39 phonemes, a word comprised of 5 phonemes would have a one hot encoded vector of 5 x 39, with 5 of those cells being 1 and all others being 0. However, not all words have 5 phonemes and our models can not deal with this variability. Therefore we also applied padding for the phoneme content. The word with the highest number of phonemes in the used data of F1 had 13 phonemes for the 3D-CNN and 15 phonemes for the ConvRNN. This meant that all other phoneme one-hot encodings had to be padded to this maximum count. We did this in the same way as we did for the videos, padding with all zeros, to the left and the right sides equally.

3.2 Model training

During training of both convolutional (3D-CNN) and RNN (ConvGRU) models we used a learning rate of 0.001. For the optimizer we used the Adam optimizer as described by Kingma and Ba (2014), and to prevent overfitting we used a weight decay of 10^{-8}. The models were implemented using Pytorch (Paszke et al., 2019) and trained on a single GPU (NVIDIA GeForce RTX 2080 Ti). We randomly divided the data into three datasets, the training set, validation set and the test set using a 80/10/10% split. The data points in the training set were used to train the model in batches of size 10. To minimize overfitting, we implemented an early stopping technique. After each epoch, i.e. when all training data points have been used to update the model's parameters, the model is validated on the data points in the validation set. If the updated model

performs better on the validation set than all other models in previous epochs, it is saved as the best performer so far. After the last training epoch is finished, the model that performed the best on the validation set at any given epoch is used for further purposes. Importantly, even though the validation set is not used to update the parameters of the model, we do use it to select the best performing model and therefore introduce bias towards it. To assess how well the model generalizes to unseen data, we compute its performance on the test set. Hence, throughout the Results and Discussion sections, when we describe model performance we refer to its performance on the test set only.

3.2.1 Custom loss function

We want our model to learn in a way that is relevant for our research. To nudge the models in the right direction, we used linguistic features of the words, more specifically the phonemic content, as proxies for the articulation information inside a custom loss function. The standard way to use an autoencoder for representation learning is to simply use the difference, the Mean Squared Error (MSE), between the input and the reconstructed output as the loss during the training phase. In addition, our custom loss function will use the phonemic content of the labels, more specifically the differences in phonemic content between words, to further adjust the parameters of the model. Formally, our custom loss function is as follows, with R being the standard reconstruction MSE, w a weight and P the custom loss component based on the phonemic Levenshtein distances between data points in the batch:

$$CustomLoss = R + w * P \tag{1}$$

The classic Levenshtein distance gives a metric of difference between two strings of letters (Levenshtein et al., 1966). It basically counts how many operations it has to take to transform the first string into the second string and returns that number as their distance. We slightly adjusted this method to work on a list of phonemes instead of a string of characters, creating the Phonemic Levenshtein Distance (PLD). For every batch during training, a phonemic Levenshtein distance matrix is generated based on the labels of the data points in the batch. Additionally a Euclidean distance matrix is generated based on the embeddings in the bottleneck layer generated during the reconstruction of each data point. Then the MSE loss is calculated between the two distance matrices, multiplied by a weight and added to the standard reconstruction loss R to create $CustomLoss$. From now on, we will refer to w as the Custom Component Weight (CCW).

3.3 Model architectures

3.3.1 2-Dimensional Convolutions

Convolutional Neural networks use a convolutional function to reduce the required number of parameters compared to fully-connected operations. In fully-

connected layers, each node is directly connected to all other nodes in both the preceding and the following layer. When using fully-connected layers with our data, connecting our input layer, which is 48 × 44 × 20 in size to only one neuron, would require 48 × 44 × 20 = 42240 connection weights. Assuming we will need a lot more than one neuron for the processing of our data, the number of parameters can quickly become infeasibly large.

Inspired by the observations of Hubel & Wiesel on the visual processing of cats and monkeys (Hubel and Wiesel, 1962, 1968), it was found that in computer vision, it is more efficient to look at local regions of an image instead of using fully-connected layers on the entire image (Fukushima and Miyake, 1982). In other words, nodes of the next layer will only get inputs from a small part of the image in the previous layer. By mapping nodes of the next layer to only small windows of nodes in the previous one, the number of parameters is greatly reduced. As an example, let's say we have an RGB image of (heigth × width × #colorchannels) = 64 × 64 × 3 in size and we want the next layer to have 32 × 32 nodes. If we use convolutions with windows of size 5 × 5, we will need (32 × 32) × (5 × 5 × 3) = 76800 parameters. If we would use a fully-connected layer, we would need (64 × 64 × 3) × (32 × 32) = 12.582.912 parameters.

By shifting a window, called the kernel, over the original input image, a convolutional function multiplies what it sees through this window with its learned weights and summarizes it into one cell of a feature map before shifting the kernel over to another part of the image to fill the next cell of the feature map. Doing this for all local areas of the image effectively creates a set of local filters. Multiple of these convolutional operations can be added on top of each other, resulting in a stack of filters for each local region, similar to how the hierarchical receptive fields are structured in the visual cortex of mammals. Each filter can extract different features which means that a convolutional layer is able to capture multiple features at a time for each local region. Formally, the value of a unit at position (x, y) in the jth feature map in the ith layer, denoted by v_{ij}^{xy}, is given by Equation 2

$$v_{ij}^{xy} = f\left(b_{ij} + \sum_{m} \sum_{p=0}^{P_i-1} \sum_{q=0}^{Q_i-1} w_{ijm}^{pq} v_{(i-1)m}^{(x+p)(y+q)}\right) \tag{2}$$

where f is an activation function, b_{ij} is the bias for that particular feature map and m indexes over the set of feature maps in the ith layer that is connected to the current feature map. P_i and Q_i are the height and width of the kernel, meaning p and q index the position within the kernel. w_{ijm}^{pq} is the value at position (p, q) of the kernel connected to the kth feature map (Ji et al., 2012).

3.3.2 3-Dimensional Convolutions

Our data consists of videos, which are 3-dimensional. In addition to the two spatial dimensions there is also a time dimension. Hence we need our convolutional model to also be able to capture dependencies on the time axis. Traditional 2D convolutions applied to videos, are not able to capture motion continuity or

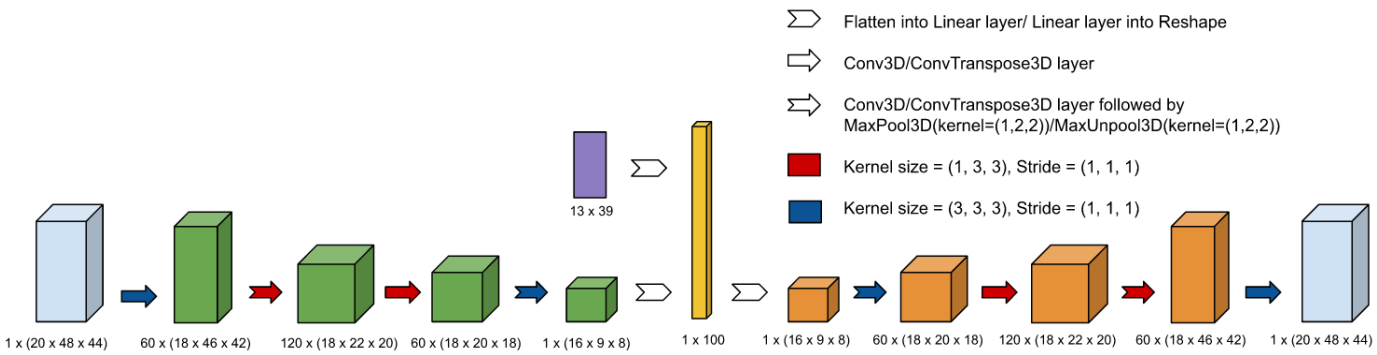


Figure 4: 3D-CNN architecture

other temporal correlations which makes them inadequate for the processing of videos (Budden et al., 2017; Tran et al., 2015). By extending the convolutional operation to a 3-dimensional convolution, the output of the convolution will preserve the temporal relations present in its input (Zhao et al., 2019; Al-Hammadi et al., 2019). A 3D convolution is performed by convolving with a 3D kernel over the 3D input data cube created when we stack images into a video. This means that the feature maps in the convolutional layer are connected to multiple contiguous frames in the previous layer, therefore also capturing dependencies over the third dimension. Formally, the value of a unit at position (x, y, z) in the jth feature map in the ith layer, denoted by v_{ij}^{xyz}, is given by Equation 3

$$v_{ij}^{xyz} = f\left(b_{ij} + \sum_{m} \sum_{p=0}^{P_i-1} \sum_{q=0}^{Q_i-1} \sum_{r=0}^{R_i-1} w_{ijm}^{pqr} v_{(i-1)m}^{(x+p)(y+q)(z+r)}\right) \tag{3}$$

where R_i is the size of the 3D kernel in the third dimension and w_{ijm}^{pqr} is the (p, q, r)th value of the kernel that is connected to the mth feature map in the previous layer (Ji et al., 2012).

3.3.3 3D-CNN AE Architecture

Our architecture is shown in Figure 4 where green represents the encoder, yellow the bottleneck layer, orange the decoder and purple the phoneme data-stream. For the encoder and decoder of the model, we took inspiration from the works of Yu et al. (2021). In addition, we used a second data stream and a linear layer to combine the two data streams into one vector. This way, we can combine the one hot encoded phonemic content with the condensed input and condense it further to a vector of arbitrary length.

The encoder condenses the input through four convolutional layers and two max pooling layers, going from dimensionality ($20\times$ 48 $\times$ 44) to (16 $\times$ 9 $\times$ 8). The output of the encoder and the phonemic data are then flattened to a 1-dimensional vector and fed into a linear layer condensing it further to a vector

of size 100. Another linear layer transforms it back into a 1 dimensional vector of size 1152 and then reshapes it to the required input size of the decoder (16 $\times$ 9 $\times$ 8). Then all operations of the encoder are "reversed" in the decoder by application of transposed convolution instead of the standard convolution. Transposed convolutions work by swapping the forward and backward passes of a standard convolution (Dumoulin and Visin, 2016). Where a standard convolution summarizes what it sees through its window of size x $\times$ y into one cell, the transposed convolution expands what it sees in the one cell towards a window of size x $\times$ y.

3.4 Convolutional Recurrent Neural Network

Recurrent Neural Networks (RNN) are networks where connections between nodes can create cycles. Because of these cycles, the derivative of each node is dependant on all earlier nodes, effectively allowing the cell to have memory (also called the hidden state of the cell). Furthermore, since this chain dependency can be arbitrarily long, a RNN allows for an arbitrarily long input sequence. This is particularly interesting for our research as it will allow us to circumvent the problem of padding. However, the longer the input sequence, the harder the model will be to train (Bengio et al., 1994). This effect is also known as the vanishing gradients problem. This problem arises due to the fact that when you apply the chain rule for the gradient calculation during training, you are multiplying small numbers (numbers between 1 and -1) with each other, resulting in even smaller numbers. As the size of the input sequence increases, and thus the number of multiplied small numbers increases, the gradients become so vanishingly small, that the model is effectively prevented from updating its weights. To combat the vanishing gradients problem, the principal of gates was proposed. These gates control what information is kept and what information gets forgotten. The most popular architectures incorporating the gated principal are the Long-Short-Term Memory cell (LSTM) (Hochreiter and Schmidhuber, 1997) and the Gated Recurrent Unit (GRU) Cho et al. (2014). After brief experimentation we decided to use the GRU in this study, given that they provide a simpler architecture, require less memory, are easier to train and some existing work indicates that GRUs may lead to better performance compared to LSTM's Amiriparian et al. (2017).

3.4.1 Gated Recurrent Unit (GRU)

The GRU was originally proposed by Cho et al. (2014). A GRU cell incorporates two gates, the update gate z_t and the reset gate r_t to control the flow of information, allowing the network to adaptively capture dependencies on different time scales. The reset gate helps capture short term dependencies and the update gate helps capture long term dependencies. The activation h_t of the GRU is defined by the following equations, where $\odot$ is an element-wise multiplication and σ a sigmoid acitvation function:

$$z_t = \sigma(W_z x_t + U_z h_{t-1}), \tag{4}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1}), \tag{5}$$

$$\tilde{h}_t = tanh(W x_t + U(r_t \odot h_{t-1})), \tag{6}$$

$$h_t = (1 - z_t)h_{t-1} + z_t \tilde{h}_t \tag{7}$$

The update gate z_t decides to what degree the unit updates its hidden state. The reset gate r_t decides when information from previous states will be forgotten. When r_t^i in a unit is close to 0, it forgets the previously computed state, effectively resetting the unit's memory making it act as if the current sequence is the first sequence it has seen. $\tilde{h}_t$ is the candidate activation, h_t is the final activation after incorporating the information from the update gate.

3.4.2 Convolutional GRU

GRUs were originally proposed for machine translation and use fully-connected layers to model the input to hidden and hidden-to-hidden transitions. In our research however, we are working with video frames, and therefore prefer convolutional operations. Combining convolutional mappings with standard GRUs becomes problematic quickly as convolutional mappings are 3D tensors, leading to an explosion in parameter numbers due to the fully-connected matrix. To combat this, Ballas et al. (2015) proposed the Convolutional GRU (ConvGRU). The ConvGRU cell is similar to the standard GRU cell but replaces the fully-connected matrix multiplications with convolutional operations. The activation h_t of the ConvGRU is defined by the following equations, where $*$ denotes a convolutional operation, $\odot$ is an element-wise multiplication and σ a sigmoid acitvation function:

$$z_t = \sigma(W_z * x_t + U_z * h_{t-1}), \tag{8}$$

$$r_t = \sigma(W_r * x_t + U_r * h_{t-1}), \tag{9}$$

$$\tilde{h}_t = tanh(W * x_t + U * (r_t \odot h_{t-1})), \tag{10}$$

$$h_t = (1 - z_t)h_{t-1} + z_t \tilde{h}_t \tag{11}$$

3.4.3 ConvGRU AE architecture

The ConvGRU architecture we used in this study is shown in Figure 5. For the overall architecture, we took inspiration from the works of Chong and Tay (2017). Similar to Chong and Tay, we used 2-layer 3D-CNN's to condense the input before feeding it into the encoding RNN layer. Contrary to Chong and Tay we used two RNN layers instead of three and used ConvGRU cells instead of ConvLSTM cells. Furthermore, after each element of an input sequence has gone through the encoding GRU cell, the hidden state of the cell which has size ($32 \times 11 \times 11$) is flattened and concatenated with the flattened one-hot encoded phonemes of the label into a 1 dimensional vector of size 4457 which

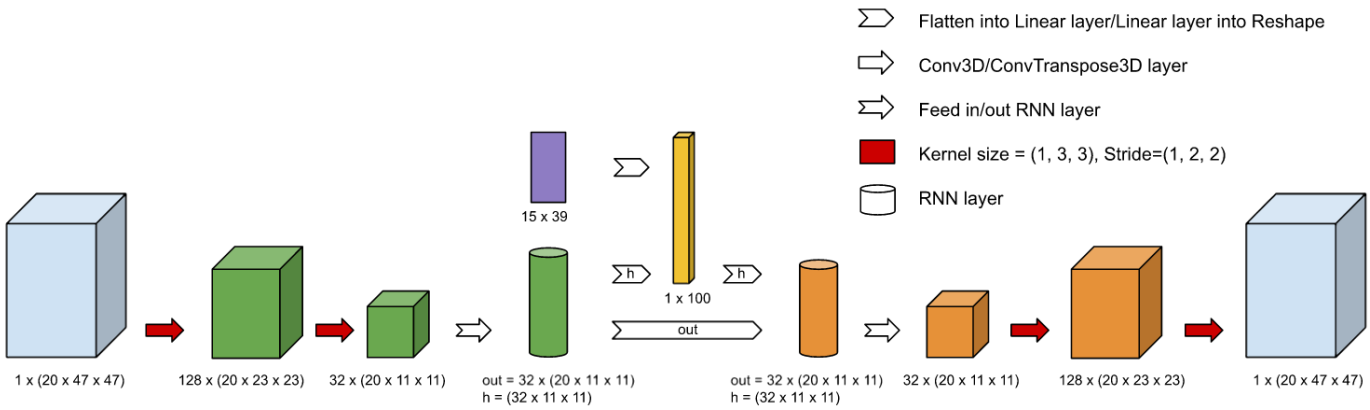


Figure 5: ConvGRU architecture

is then used as input to a linear layer that condenses it to a vector of length 100. This vector is then brought back to a vector of size 3872 after which it is reshaped to the size of the hidden state of the decoding GRU cell ($32\times$ 11 $\times$ 11). Thus, the hidden state of the last time point of the encoding GRU cell is condensed into the bottleneck representation and then reshaped back to be the hidden state of the decoding GRU cell at the first time point.

3.4.4 Batching

We trained our models in batches of 10. Training in batches means that the gradients are computed over a small set of data points rather than for every data point separately. Not only does this speed up the training process, taking multiple data points into account during computation may help to smooth out the gradient (Breuel, 2015).

The main advantage of a RNN is that it can deal with varying length inputs. However, when training the model in batches, the items within a batch are still required to be of the same size. This problem can be dealt with in three ways: (1) padding, (2) using a batch size of 1 or (3) forcing equal-sized batches. Since we chose to use a RNN to escape the costs of padding, we decided against using padding. Using a batch size of 1 effectively means that the model updates its weights after every training instance. Not only does this make training a lot slower, it prevents us from using our custom loss component. The custom loss component requires the batch size to be larger than 1 since it compares the relative distances between the instances of a batch. Therefore, we went with the third option, forcing equal-sized batches.

By forcing equal-sized batches, we mean enforcing that whenever one item in a batch is of size x, then all other items also have to be of size x. We did this by dividing the data into groups based on their sequence length. During training, batches are extracted from a random group (without replacement). When all of the instances of a group are used up, it will be flagged as empty and will not be

a candidate for subsequent extraction. When all groups are empty, the epoch is finished and the groups are reinitialised and randomly shuffled within groups.

3.4.5 Parameter optimization

After training a 3D-CNN or ConvGRU model, we input all the words in our datasets (train, validation and test sets), to extract the embeddings from the bottleneck layer. Before clustering the embeddings, we need to determine their quality. Our models effectively have two target functions. The first is the reconstruction of the original video. It is easy to quantify the error between the original image and the reconstructed image, allowing active penalization for any deviations from that function. In contrast, there is no straightforward metric to asses the quality of the learned embeddings. We want our model to learn representative feature vectors for the articulation pattern of each word. Without a quantifiable assessment metric, we can not actively penalize the model for deviating from this target. Not having a ground truth to compare the learned embeddings with, makes it difficult to know whether the model actually learned relevant embeddings or not, so the best thing we can do is approximate the quality by correlating the embeddings with proxy metrics. We used two different proxy functions: the PLD matrices and the EMA matrices.

First, we generate Euclidean distance matrices for each group of identical syllable counts based on the embeddings generated by the model (every group should only contain members with the same number of syllables due to how the EMA matrices are structured). Then we compute the correlation of these Euclidean distance matrices per syllable count with the corresponding PLD and EMA distance matrices. The CCW that produced the model with the best correlations, and is thus expected to have the highest quality embeddings, will then be used for clustering.

We use the correlation with the PLD distance matrix because we have incorporated the Phonemic Levenshtein information in the custom loss function. By correlating the embeddings with the PLD matrices, we can get an insight in the extend to which the phonemic information has been incorporated in the embeddings. A very low correlation would mean that the embeddings do not represent any phonemic information. A very high correlation would mean that the embeddings effectively mirror the phonemic information. We are looking for correlation values somewhere in between as low correlations mean that the addition of the phonemic information has been redundant and high correlations mean that we are basically using the phonemic levenshtein matrix as embedding, thus making the autoencoder redundant. The PLD correlation metric also gives us an insight in how different values for CCW influence the generated embeddings. If the PLD correlation is too low we would want to increase the weight, and if the PLD correlation is too high we would want to lower the weight.

Secondly, we need to make sure that the learned embeddings not only capture linguistical features, but also capture meaningful information about the words articulation. For this we evaluated how well the learned embeddings correlated with the articulation data collected with EMA. The EMA matrices are

provided by a colleague who performed clustering of articulation patterns using Electromagnetic Articulography (EMA) data, instead of MRI. We use these correlations as an approximation of how close the difference between learned embeddings is to the difference in EMA profiles across words. Taken together, we are looking for model embeddings that have intermediate levels of correlation with the PLD matrices and high correlation with the EMA matrices.

We will start training both architectures with a CCW of 0, giving us the embeddings generated by the vanilla MSE loss. Then we start training the architectures with a CCW of 1.0E-6. If we don't see significant PLD correlations we increase the CCW by a factor of 10 and try again. Seeing an increase in PLD correlation indicates that the custom component of the loss function is starting to have an effect. From the point that we see an increase in PLD correlation we start increasing the CCW with smaller increments until the EMA correlations start to drop.

3.5 Cross participant transferability

Due to time restraints, it is not feasible to perform an in depth parameter optimization for each participant separately. Therefore, we will do the optimization steps discussed in the previous section for participant F1 only. When we have determined the optimal parameters for F1, we investigate the degree of transferability of the model to the other participants. We will do this by first testing the model trained on the F1 training set on the test sets of the other participants. This will give us an idea of how well the model generalizes to unseen facial structures.

It is hard to gather as much words as Narayanan et al. (2014). Therefore, it would a valuable asset if a pre-trained model would only need a small set of words to be fine-tuned for a new participant. Therefore, we will investigate how many words are needed to fine-tune our model to a new participant. We will do this by fine-tuning the best performing model M that was trained and tested on the train set tr^{f1} and test set te^{f1} of participant F1, on train sets of increasing size of the other participants tr^{o}. First, we train M on tr^{o} for 20 epochs and then the fine-tuned model is tested on the test set of the other participant te^{o}. We start with a fine-tune set of size 100 and iteratively increase it by 100 until it is of size 500.

3.6 Embedding space quality

To get an insight in the quality of the embedding space itself we will calculate some properties of the space: the Average Duplicate Distance (ADD), Average Duplicate Ending Distance (ADED) and Average Duplicate Beginning Distance (ADBD). The ADD represents the average Euclidean distance between duplicate labels within the embedding space. As duplicate labels represent multiple articulations of the same word, these data points should be close to each other in the embedded space. The ADED represents the average Euclidean distance between labels that end with the same 2 phonemes and the ADBD represents the

average Euclidean distance between labels that begin with the same 2 phonemes. We then divide these metrics with the Average Non-duplicate Distance (AND), which represents the average distance between any non-duplicate pair, to scale our metric values compared to the non duplicate words. Through this comparison we can get an insight in how similar words are distributed over the space compared to dissimilar ones.

3.7 Clustering

Dividing a set of data points into a set of clusters is a difficult computational problem. Inspecting every single cluster combination quickly becomes infeasible as the number of data points increases. Over the years, many cluster algorithms have been proposed that avoid using brute force techniques in order to save computational time. For our research we will use the K-means algorithm.

The K-means algorithm partitions N objects, each having P features into K classes $(C_1, ..., C_K)$ where C_k is the set of n_k objects in cluster k. To avoid using brute force, the K-means algorithm uses an iterative approach in which it tries to partition the data so that the squared Euclidean distance between the row vector for any data point and the centroid vector of its respective cluster is at least as small as the distances to the centroids of the remaining clusters (Steinley, 2006). The centroid of a cluster c_k is found by averaging each variable over the objects within the cluster. E.g., the centroid value $\bar{x}_j^{(k)}$ is given by:

$$\bar{x}_j^{(k)} = \frac{1}{n_k} \sum_{i \in C_k} x_{ij} \tag{12}$$

The K-means algorithm finds the clusters in the following 4 iterative steps:

1. K initial seeds (S^1, ..., S^K) are defined by P-dimensional vectors (s_0^k, ..., s_P^k) and the squared Euclidean distance between the ith object and the kth seed vector, $d^2(i,k)$ is given by:

$$d^2(i,k) = \sum_{j=1}^{P} (x_{ij} - s_j^{(k)})^2 \tag{13}$$

   Each object is allocated to the cluster for which $d^2(i,k)$ is the lowest.

2. After the initial object allocation, the cluster centroids are obtained with Equation 12. Then each object is moved to the cluster which centroid is closest (using $d^2(i,k)$).

3. Cluster centroids are recalculated with the updated set of members.

4. Step 2 and 3 are repeated until no object can be moved between clusters anymore.

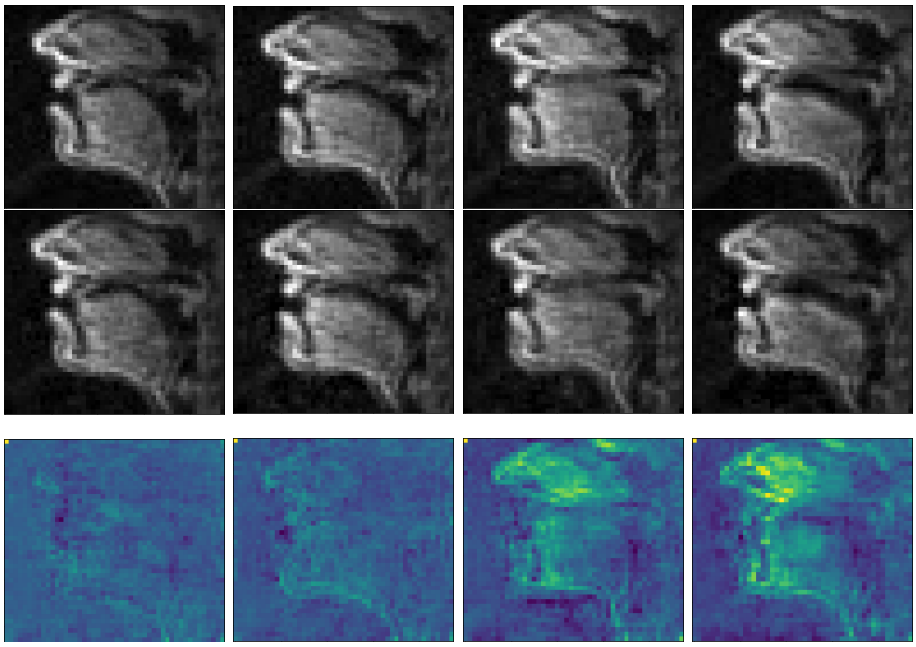
After generating the clusters, we find the center point of the clusters in the embedding space. Than we find the data point that is closest to the centre and mark this point as the representative of the cluster. Since we want to find the 20 most distinct words we used $K = 20$, resulting in 20 clusters.

3.8 Assessing cluster quality

As mentioned in the previous section, a set of data points can be clustered in many different ways. In our case, we want the clusters to represent similarities in articulation patterns. To evaluate our clusters we introduce the following metrics: the Separated Duplicates (SD), Average Character Count Difference (ACCD) and Average Levenshtein Difference (ALD) within and outside of the clusters. The SD score represents the percentage of duplicates that were not assigned to the same cluster. As duplicate words have the same articulation pattern, we want them to be clustered in the same cluster and thus we want the SD score to be as low as possible.

The ACCD represents to what extend words of similar length are grouped together. The ACCD is computed as follows: for each cluster c, the character count difference is calculated between every member m_c and the representative of the cluster r_c, and the average is taken. Then, for every member of every other cluster m_o, the character count difference is calculated compared to r_c. We do this for each cluster resulting in 2 lists of 20 average character count differences, one for the members within the clusters and one for the members outside the clusters. Then we apply a Wilcoxon test between the two lists. A Wilcoxon test is used to compare two groups and see whether they are significantly different from each other (Wilcoxon, 1945). The ACCD score will give us an insight in how words with different lengths are distributed over the clusters. If the within ACCD score is lower than the outside ACCD score, we know that members within a cluster are more similar in size than those outside of the cluster, indicating that the clustering has taken word length into account.

In a similar way we will compute the ALD score. The ALD score represents to what extend words with similar phoneme content are grouped together. For each cluster c, the phonemic Levenshtein difference is calculated between every member m_c and the representative of the cluster r_c, and the average is taken. We again do this for each cluster and apply a Wilcoxon test. The ALD score will give us an insight in how similar words in their phonemic content are distributed over the clusters. Similar to the ACCD score, a low within ALD score compared to the outside ALD score will indicate that words within a cluster are phonemically more similar to those outside of the cluster, indicating that the clustering has taken the phonemic content into account.



(a) 40 loss (b) 100 loss (c) 300 loss (d) 500 loss

Figure 6: Examples of some reconstructed frames for participant F1 with different accuracy's on the test set. The top row represents the original input, the middle row represents the corresponding reconstructed output. The bottom row represents the difference between the top and middle row per pixel

4 Results

4.1 Reconstruction accuracy

Both the 3D-CNN and the ConvGRU models were able to reconstruct well. The 3D-CNN took around 60 minutes to train 100 epochs, and the Conv-GRU about 45 minutes. The best performing 3D-CNN model for participant F1 achieved an average reconstruction loss of 28 on the test set and the best performing ConvGRU model a reconstruction loss of 7. The addition of the custom loss component did not have a large impact on the reconstructive capabilities of the models. The test loss fluctuated within a range of 30 points for different CCW's for both the 3D-CNN and ConvGRU. This means that all models reported in this section are well within an acceptable range of reconstructive accuracy (see Figure 6 for reference to what different loss values imply)

Thus, the ConvGRU model achieved a test loss that was on average four times lower than the 3D-CNN. On top of this, the test loss for the 3D-CNN models are calculated over both padded and non-padded frames. Since padded

frames, are generally easier to reconstruct, the reconstruction loss on only the non-padded input frames will be slightly higher than what is reported for the 3D-CNN. As the ConvGRU does not use padding, it does not suffer from this phenomenon. Therefore, the gap in reconstructive performance between the two architectures is slightly higher than the numbers would suggest.

4.2 Parameter optimization

	Weight: 0 Test loss: 28			Weight: 0.1 Test loss: 55			Weight: 0.5 Test loss: 42			Weight: 1 Test loss: 50		
	syl	r	p	syl	r	p	syl	r	p	syl	r	p
	1	0.135	0	1	0.149	0	1	0.381	0	1	0.632	0
	2	0.094	0	2	0.131	0	2	0.709	0	2	0.883	0
PLD	3	0.052	3.8E-18	3	0.180	7.7E-205	3	0.561	0	3	0.720	0
	4	-0.031	0.244	4	0.054	0.043	4	0.367	3.9E-45	4	0.573	6.8E-121
	5	0.004	0.970	5	0.175	0.125	5	0.421	1.2E-4	5	0.576	3.4E-8
	1	0.042	3.2E-41	1	0.098	2.7E-193	1	0.121	1.8E-296	1	0.123	2.3E-305
	2	0.077	3.0E-88	2	0.087	2.2E-102	2	0.068	4.1E-64	2	0.0457	1.0E-29
EMA	3	0.015	0.164	3	0.002	0.868	3	0.034	2.3E-3	3	0.079	2.1E-12
	4	0.046	0.327	4	0.048	0.351	4	0.182	3.7E-4	4	0.096	0.061
	5	0.126	0.464	5	0.186	0.278	5	0.427	9.4E-3	5	0.711	1.0E-6
	Weight: 1.5 Test loss: 64			Weight: 2 Test loss: 60			Weight: 5 Test loss: 48			Weight: 10 Test loss: 43		
	syl	r	p	syl	r	p	syl	r	p	syl	r	p
	1	0.641	0	1	0.611	0	1	0.957	0	1	0.975	0
	2	0.866	0	2	0.840	0	2	0.646	0	2	0.967	0
PLD	3	0.759	0	3	0.690	0	3	0.830	0	3	0.905	0
	4	0.590	7.7E-130	4	0.445	3.9E-68	4	0.666	1.4E-177	4	0.798	3.7E-305
	5	0.654	8.4E-11	5	0.350	1.6E-3	5	0.397	3.2E-4	5	0.610	3.0E-9
	1	0.103	6.4E-215	1	0.108	2.0E-239	1	0.081	4.8E-132	1	0.068	2.7E-95
	2	0.049	2.4E-34	2	0.056	2.2E-44	2	0.014	4.2E-132	2	0.025	4.3E-10
EMA	3	0.039	4.6E-4	3	0.052	3.0E-6	3	0.037	1.1E-3	3	0.048	1.8E-5
	4	0.094	6.7E-2	4	0.189	2.2E-4	4	0.080	0.121	4	0.042	0.413
	5	0.570	2.8E-4	5	0.531	8.5E-4	5	0.578	2.2E-4	5	0.429	0.009

Table 1: Correlation values for the embeddings produced by the 3D-CNN model, trained with different CCW's, correlated with the PLD and EMA distance matrices

Table 1 and 2 show the results of the embedding correlations for the 3D-CNN and ConvGRU model respectively. Each model was trained with a different CCW and correlated to both the PLD and EMA distance matrices. An increase in PLD correlation can be observed as we increase the CCW for both the 3D-CNN and the ConvGRU. However for the 3D-CNN the correlations seem to decrease again after the CCW increases from 1 and for the ConvGRU when the CCW increases from 0.006.

None of the 3D-CNN and ConvGRU models produced significant correlations with the EMA data for words that have less than 5 syllables. For the 5-syllable words however, we do observe significant correlations. For the 3D-CNN, when

	Weight: 0 Test loss: 30			Weight: 0.0001 Test loss: 11			Weight: 0.001 Test loss: 7			Weight: 0.002 Test loss: 7		
	syl	r	p	syl	r	p	syl	r	p	syl	r	p
	1	0.004	0.002	1	0.170	0	1	0.884	0	1	0.668	0
	2	-0.009	5.5E-5	2	0.531	0	2	0.913	0	2	0.853	0
PLD	3	-0.007	0.187	3	0.569	0	3	0.831	0	3	0.812	0
	4	-0.036	0.091	4	0.442	1.8E-106	4	0.714	0	4	0.732	0
	5	-0.075	0.384	5	0.262	0.002	5	0.512	1.8E-10	5	0.558	1.8E-12
	1	0.028	1.2E-17	1	0.011	0.001	1	0.037	1.6E-29	1	0.025	3.9E-14
	2	0.006	0.136	2	0.011	0.004	2	0.012	2.7E-3	2	0.021	1.0E-7
EMA	3	0.015	0.163	3	0.018	0.099	3	0.041	1.8E-4	3	0.047	2.0E-5
	4	0.005	0.914	4	-0.005	0.916	4	-0.018	0.694	4	-0.008	0.853
	5	-0.114	0.405	5	-0.082	0.554	5	0.202	0.139	5	0.292	0.031
	Weight: 0.004 Test loss: 12			Weight: 0.006 Test loss: 10			Weight: 0.008 Test loss: 17			Weight: 0.1 Test loss: 12		
	syl	r	p	syl	r	p	syl	r	p	syl	r	p
	1	0.918	0	1	0.878	0	1	0.852	0	1	0.476	0
	2	0.937	0	2	0.923	0	2	0.903	0	2	0.693	0
PLD	3	0.826	0	3	0.839	0	3	0.816	0	3	0.748	0
	4	0.738	0	4	0.767	0	4	0.716	0	4	0.703	0
	5	0.737	1.4E-24	5	0.560	1.2E-14	5	0.670	4.6E-19	5	0.608	4.1E-15
	1	0.042	1.4E-36	1	0.018	8.9E-8	1	0.018	4.8E-8	1	0.017	2.0E-7
	2	0.013	8.4E-4	2	0.024	1.8E-9	2	0.009	0.003	2	0.01	0.01
EMA	3	0.043	7.9E-5	3	0.033	0.002	3	0.02	0.007	3	0.036	9.5E-4
	4	-0.011	0.802	4	-0.004	0.933	4	-0.02	0.662	4	-0.04	0.417
	5	0.302	0.025	5	0.4	0.025	5	0.261	0.054	5	0.238	0.08

Table 2: Correlation values for the embeddings produced by the ConvGRU model, trained with different CCW's, correlated with the PLD and EMA distance matrices

the CCW is 0 and thus not used, no significant correlation was found between the embeddings and the EMA data (r=0.126, p=0.464) However, by adding the custom component to the loss function, this correlation increases to a highly significant correlation, in the case of CCW = 1 (r=0.711, p=1.0E-6). From there it goes back down again to a lower but still significant correlation of (r=0.429, p=0.009) for CCW = 10. For the ConvGRU, the EMA correlations also increase from no significant correlation at CCW = 0 (r=-0.114, p=0.405) to a significant correlation of (r=0.4, p=0.025) at CCW = 0.006. For higher weights the correlations decrease again. Based on these results we decided to use the following weights for the models used for the rest of the results: 3D-CNN model with weight = 1 and ConvGRU model with weight = 0.006.

4.3 Cross participant transferability

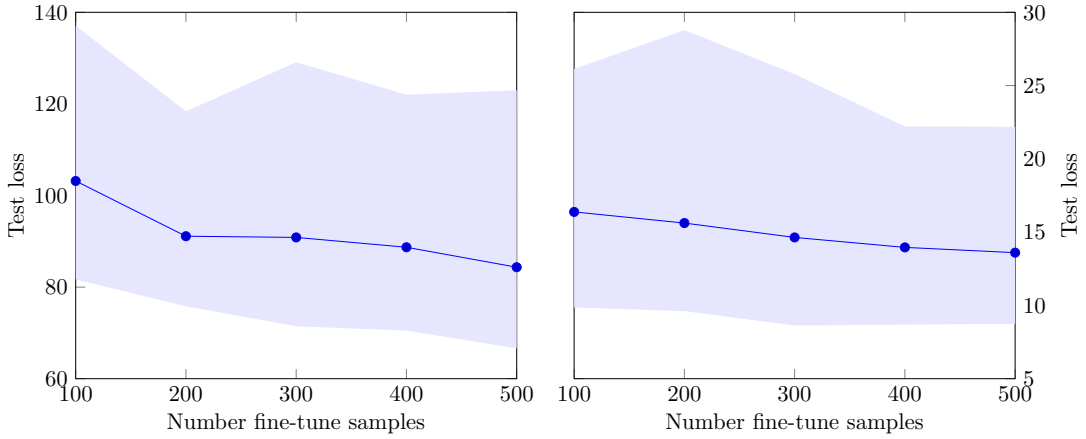
So far, all the results were gathered on the data of participant F1 only. Now we will show the results of the fine-tune experiments for the other participants. Table 3 shows the results for the F1 optimized 3D-CNN and ConvGRU model tested on all other participants.

For the 3D-CNN model, the initial test losses, thus without fine-tuning, on

	F2	F3	F4	F5	M1	M2	M3	M4	M5
3D-CNN									
Without fine-tuning	370.41	403.26	309.81	273.92	530.79	450.91	388.76	346.84	640.66
After fine-tuning	82.79	88.55	80.43	72.3	66.63	76.74	84.57	84.1	122.98
ConvGRU									
Without fine-tuning	58.37	77.17	51.06	54.09	49.03	82.51	60.45	46.10	101.56
After fine-tuning	14.37	16.74	12.79	11.19	8.78	13.79	10.88	12.03	22.23

Table 3: Test losses of models trained on F1, tested on other participants, before and after fine-tuning with fine-tune set size = 500

the test sets of the other participants were quite poor with the best performance seen on participant F5 (273.92) and the worst on M5 (640.66). However, fine-tuning the model on small sets of data significantly improved performance. Although for none of the participants the model achieved the same test loss as it did on participant F1 (50), the performance did get below a loss of 90 for 8 out of 9 participants with participant M1 performing the best (66.63) and M5 the worst (122.98). For the ConvGRU we see that the initial test losses are a lot lower than was the case for the 3D-CNN, with the best performance seen on participant M4 (46.10) and the worst, again, on M5 (101.56).



(a) 3D-CNN

(b) Conv-GRU

Figure 7: Test losses for the model originally trained on F1 and fine-tuned on the other participants with increasing fine-tune set sizes

Therefore, the GRU model, on average, generalizes better across participant without fine-tuning than the fine-tuned 3D-CNN models. After fine-tuning the GRU models, 4 out of 9 participants achieved a test loss within 20 percent of the original test loss on F1 (10) and one participant performed even better than that (participant M1). The other participants did not get within 20 percent of the original test loss but did get close with the worst being participant M5

again with a test loss of only 22.23.

Figure 7a and 7b show the intermediate fine-tuning results for the 3D-CNN and ConvGRU respectively. The blue line shows the mean values over all participants and the shade borders show the maximum and minimum values. The first thing to note is that the y-axis is of different scales. Not only did the ConvGRU model achieve high performance with a fine-tune set size of 500, it already showed good results with a fine-tune set size of 100. The addition of 400 extra fine-tune samples only decreased the mean loss with a few points. For the 3D-CNN the difference in performance between 100 and 500 fine-tune samples is a lot bigger, indicating that the 3D-CNN model requires a larger fine-tune set size than the Conv-GRU.

4.4 Quality of the embedding space

	F1	Mean	Median	Std	Min	Max
3D-CNN						
ADD	0.58	0.59	0.60	0.03	0.54	0.64
ADBD	0.97	0.96	0.96	0.01	0.95	0.97
ADED	1.03	1.02	1.02	0.004	1.01	1.03
ConvGRU						
ADD	0.39	0.49	0.50	0.02	0.46	0.51
ADBD	0.94	0.95	0.94	0.01	0.94	9.96
ADED	0.99	0.99	0.99	0.01	0.96	1.00

Table 4: Quality metrics of the embedding space

To get a better idea of the quality of the embedding space, we calculated the ADD, ADED and ADBD, which are shown in Table 4. As the parameters were optimized for F1, we show the results for the F1 model separately. The other metrics are the mean, median and standard deviations for the metrics across all the fine-tuned models for each of the 10 participants. For the 3D-CNN and ConvGRU, the average distance between a duplicate pair is around twice as small as the average distance between a non-duplicate pair. This means that words with the same label lay twice as close together in the embedded space compared to words with differing labels. A similar but smaller relation is observed for the ADBD, meaning that words with similar beginnings lay slightly closer to each other than words with different beginnings. The ADED score tells us that words that have similar endings do not lay closer to each other than words with different endings. One more thing to not is that the devation across participants is very low, indicating that the quality of the embedding space generalizes over participants.

4.5 Clusters

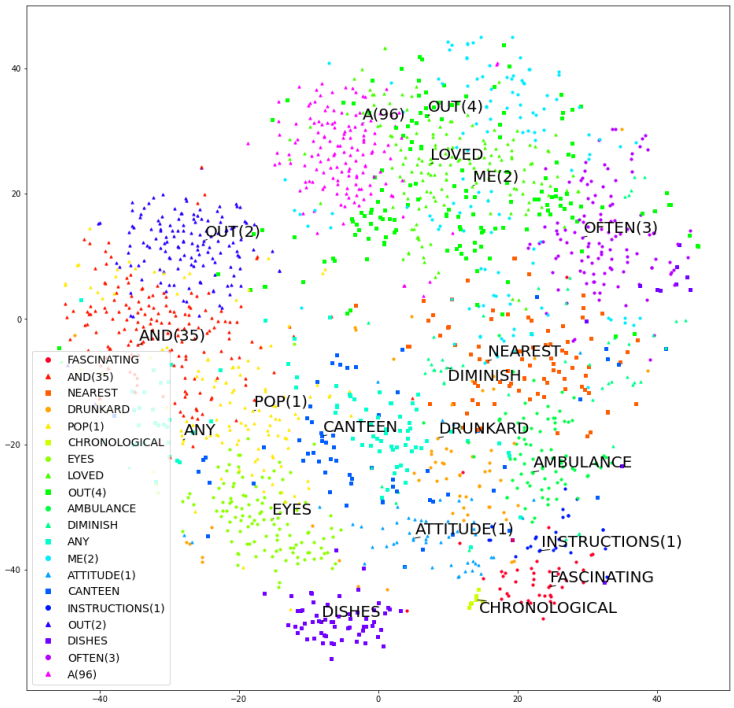
We used the t-SNE algorithm following the guidelines set by Wattenberg et al. (2016) to create a 2-dimensional visualization of the 100-dimensional embedding space generated by the models trained on F1. In this 2-dimensional space, we coloured the data points based on what cluster they were assigned to by the Kmeans algorithm and labelled the cluster representatives. The results of the 3D-CNN and ConvGRU are shown in Figure 8a and 8b respectively.
We can see how the data points do not form perfect clusters but that the color assignment is not random either. Additionally the cluster representatives can be seen in the legends of the two plots. We can see how the clusters generated by the 3D-CNN model produce one duplicate representative pair (OUT(2) and OUT(4)). The clusters generated by the ConvGRU do not contain a duplicate representative pair. Because the t-SNE algorithm makes various adjustments to enable the mapping of the high-dimensional space to a 2D representation, this 2D-representation should be interpreted with caution. Therefore, we additionally provide the SD, ACCD and ALD score in Table 5.

	Score	Wilcoxon statistic	Wilcoxon p
3D-CNN			
SD	0.293	-	-
ACCD	0.503	9	6.2E-5
ALD	0.989	70	0.202
ConvGRU			
SD	0.106	-	-
ACCD	0.503	6	2.7E-5
ALD	0.946	32	0.004

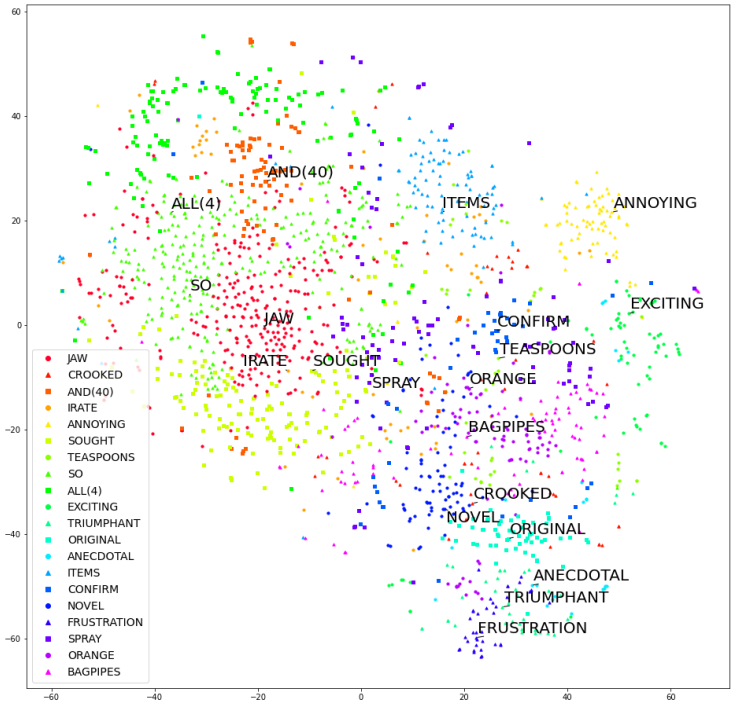
Table 5: Quality metrics of the clusters generated by the two models

The SD score is a value between 0 and 1 which can be interpreted as the percentage of separated duplicate pairs out of all duplicate pairs. The ACCD and ALD scores are reported as the within score divided by the outside score. Therefore, a value below 1 indicates a smaller difference between members within the same cluster compared to those outside and a value above 1 indicates a larger difference between members of the same cluster compared to outside. Additionally the results from the Wilcoxon test are shown for both the ACCD and ALD comparisons between within cluster and outside cluster averages. The results show that the clusters generated by the ConvGRU model outperform the 3D-CNN. The SD score of the 3D-CNN shows that almost 30 percent of the duplicate pairs are seperated in different clusters compared to only 11 percent for the ConvGRU.

The ALD score for the 3D-CNN is only slightly lower than 1 indicating that the phonemic content of the words are distributed across clusters instead of within. This is further shown by the results of the Wilcoxon test which, with a p-value of 0.202, does not produce a p-value below 0.05 meaning we cannot reject



(a) 3D-CNN cluster space



(b) ConvGRU cluster space

Figure 8: t-SNE visualization of the embedding space for the 3D-CNN and ConvGRU respectively. Data points are coloured based on their cluster membership and cluster representatives are labelled

the null-hypothesis that the phonemic Levenshtein distances between clusters is equal. For the Conv-GRU we see a lower ALD score of 0.946 which indicates a higher seperation of phonemic content across clusters compared to the 3D-CNN. This is further shown by the results of the Wilcoxon test which results in a p-value of 0.004 which is lower than 0.05 meaning we can accept the alternative hypothesis that the phonemic Levenshtein distances between clusters are not equal in a statistically significant way.

The ACCD score is equal for both models at 0.503 which is a lot lower than 1. This indicates that words within a cluster are on average twice as close to each other in length as words outside of their cluster and therefore indicates that word length is a big separating factor (this can also be observed in the visualizations of Figure 8a and 8b). The results of the Wilcoxon tests further show that this relation is highly significant in both the 3D-CNN and Conv-GRU with values of 6.2E-5 and 2.7E-5 respectively.

5 Discussion

We have seen that a ConvGRU autoencoder architecture creates a higher quality embedding space and higher quality cluster space and is therefore better suited for the purpose of creating word embeddings than a 3D-CNN architecture. Therefore, going forward, convolutional recurrent models should be used over solely convolutional ones when generating embeddings from spoken word sequences. Furthermore, we have seen that the combination of a second datastream of phonemic content and a custom loss function incorporating the relative PLD of the items in each batch, resulted in a more interpretable embedding space. Model performance did not differ substantially across participants, suggesting high generalizability across healthy people, which further suggests a generalizability to people with LIS. We found 20 clusters in the embedding space of participant F1, and saw how the representatives did not include duplicates or multiple similar words and were ordered on length across the embedding space. Therefore, for further research on direct word encoding using BCIs, we recommend using the representatives of the 20 clusters generated by the ConvGRU model trained on participant F1 which can be found in Table 6. Our findings suggest that these 20 words should provide BCI applications the most reliable decoding while still providing 20 degrees of freedom for the user.

5.1 Data quality

The transcriptions included in the USC-TIMIT dataset unfortunately suffered from accuracy problems. During our research we primarily focused on the transcriptions of participant F1, and to the best of our guess we approximate that an error occurred in the transcription of between 2 to 5% of the words. This means that some words are cut off early, some words are cut off late and some words have a completely different label. Furthermore, from quick inspection of the other participants' transcriptions, we found similar issues and therefore

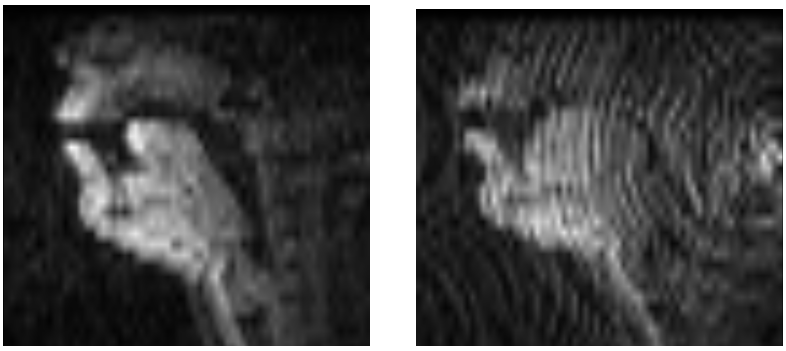


Figure 9: Example of a clean frame (left) and noisy frame (right) of participant M5

expect this to be a dataset wide problem. We did make an effort to create our own transcriptions, but without luck. Unfortunately, our transcriptions also suffered from mistakes in around the same percentage range as the original transcriptions. This was probably caused by the poor quality of the sound files, which were recorded within the noisy environment of an MRI machine. Therefore we decided to go with the original transcriptions and accept the 2-5% noise. Additionally to the transcription noise, some of the participants included noisy capture quality, especially participant M5. As can be seen in Figure 9, occasionally the frames of a video would contain strange circular patterns. This could explain why the models fine tuned on participant M5 produced such poor results compared to the other participants. Therefore, for further research we would suggest using transcriptions that were manually transcribed making sure that every word contains all the appropriate frames and no more than that. Furthermore, we would suggest further research using this dataset to exclude the data of participant M5 due to the reduced capture quality.

For further research on direct word decoding, we would suggest using MRI videos of single spoken words only. The data used for this research consisted of broken down sentences which will be effected by coarticulation. Coarticulation arises when multiple words are spoken in quick succession where the endings of the previous word and beginning of the next somewhat morph in to each other during articulation. This can result in the articulated phonemes not exactly corresponding to the phonemic content of the words. Therefore, there is an expected factor of noise between the phonemic datastream, where the phonemic content was determined on the string of the word only, and the actual articulated phonemes by the speaker. Moreover, the ADBD and ADED scores could potentially have been affected by coarticulation. It is hard to quantify the exact effect coarticulation has had on the word embeddings, but to exclude it all together, we advise further research to use data of single spoken words only.

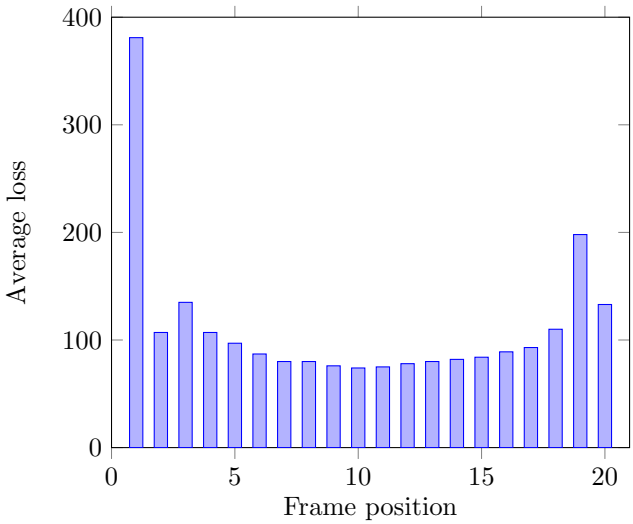


Figure 10: Average contribution to loss per frame position for the 3D CNN model trained with CCW = 1

5.2 3D-CNN vs. ConvGRU

We have seen that both deep learning architectures were suitable for use in an autoencoder framework to condense rtMRI videos of word articulation into a representative vector of 100 values and then reconstruct it back into the original input with acceptable accuracy. However, we have seen that purely on reconstruction accuracy, the ConvGRU achieved a loss 4 times as low as the 3D-CNN. Furthermore, we have seen that, following our proposed metrics, the quality of the cluster space was higher for the ConvGRU compared to the 3D-CNN. The most likely explanation for the underperformance of the 3D-CNN model is the necessity of padding. Although a necessity, this padding comes with a cost. This cost is illustrated in Figure 10. This figure shows the average reconstruction error for each frame position of a video. We can see a concave pattern in the average loss contribution for each frame position. This can be explained by the fact that the videos were padded from the centre outward, so the more left/right a frame is, the more often this frame will have been padded. It is worth noting that the average is taken only over the frames that were not padded i.e. when a video only had 14 frames, the three left most and three right most frames were excluded from the calculation of the average. The big spike in the first position might be explained by the fact that whenever a video had an odd amount of frames, the left was padded one more than the right. Therefore, videos with 19 frames would only have a padded frame on the left side. Because we see this concave pattern, short sequences will have a higher reconstruction accuracy than longer sequences. This means there is a performance bias for shorter sequences which could be the explanation for why the

3D-CNN performed worse than the ConvGRU.

5.3 Phonemic information

We have seen that adding a second datastream of phonemic content and a phonemic Levenshtein custom component to the standard loss function had a significant positive effect on the learned embeddings. We saw that without the custom loss component, the models learned embeddings that did not correlate with either phonemic information (PLD), nor articulatory information (EMA). By adding the custom component, and thus adding PLD information, we did not only see an increase in PLD correlation, but also in EMA correlation. This suggests that adding phonemic information to the model made it produce embeddings that better resembled the articulation patterns. On top of the custom loss component, the phonemic datastream was also of importance. During experimentation without this second datastream, some correlation could be observed between the embeddings and the PLD but no model achieved higher correlations than r=0.2 on the PLD matrices and no model achieved any significant correlation with the EMA matrices.

This indicates that the model needs both the phonemic datastream and the loss function to produce the right embeddings. This could be explained by the fact that although both additions provide the same phonemic information, they do it in different ways. The phonemic datastream provides information about the phonemic content of the word itself. The custom loss component provides phonemic information relative to other words. Therefore, the combination of the two allows for each data object (word) to have knowledge about what phonemic content it possesses itself and how that content relates to the content of the data objects around it.

5.4 Embedding space

Despite the embeddings of the 3D-CNN producing higher correlations with the EMA data, words with the same label were on average farther apart than those in the ConvGRU embedding space, but not by much. Furthermore, words with identical beginnings (ADBD) and identical endings (ADED) lay on average almost equally far apart in the embedding spaces of both architectures. Therefore, it is hard to determine which architecture produced the highest quality of embeddings. Looking only at the participant which was optimized, we can say that the ConvGRU model outperformed the 3D-CNN model. However when we take the fine-tuned models of all other participants into account the two architectures seem to generate embeddings of similar quality.

Furthermore the results of the ADBD and ADED seem to suggest that the last part of a word has less influence on the differentiability between two words than the first part of a word. For both the 3D-CNN and ConvGRU, the ADBD is lower than the ADED. This means that words with similar beginnings are closer to each other in the embedding space than words that have similar endings. Although the differences are not large, they could suggest an interesting

relation between articulation differentiability and character position within the word, namely that the characters at the beginning of a word provide more characteristic features than those at the end. This could potentially be of use in the decision process of what words to use in a BCI application as these findings suggest that the focus should be more on words with distinct beginnings than distinct endings. However, more research is needed to determine the extend of this relation.

5.5 Generalizability

We have seen that the models used, especially the ConvGRU architecture, have a high degree of transferability across different participants, both within and outside of the same gender. We saw how the ConvGRU model trained on the train set of participant F1 was able to achieve good test accuracy's on test sets of other participants without any fine-tuning despite the differences in their facial anatomy (see Figure 2). Furthermore, we saw that it only takes a small amount of data to fine-tune the Conv-GRU model to achieve similar performance on other participant as it did on the test set of the same participant that it was trained on. In practice this will mean that this approach can easily be applied on multiple different participants. Instead of having to gather thousands of words to train a new model for every participant, we only need to have a pre-trained model and gather a small set of words per participant (around 100 samples). Therefore, the neural network approach covered in this research can provide an important addition to the articulation pattern research. Where the clustering of EMA data suffers from the necessity of padding and a lack of transferability across participants, the ConvGRU approach can provide a useful alternative as it does not suffer from these issues.

One disadvantage of the ConvGRU model however, is the phonemic information datastream. This second datastream of phonemic content is hard coded to the English language. The high generalizability across participants shows promise to be generalizable across languages, however this is made difficult by the phoneme data stream as it incorporates the one hot encoding of only the 39 phonemes present in English. Other languages have different numbers of phonemes, Dutch for example only has 35. However, Dutch also includes phonemes that are not present in English. Therefore the one hot encoding of the English phonemes can not simply be reused for a language such as Dutch by padding the phonemes that are present in English but not in Dutch. The phonemes that are present in Dutch but not in English also have to get their spot in the one hot encoding. Thus, although the ConvGRU approach generalizes well over multiple people within the same language, to know whether it can generalize across languages further research has to be done. A possible solution could be to enlarge the one-hot encoded vector to be large enough to accommodate the largest number of phonemes present in all languages of interest and use padding for the languages that possess less phonemes. Another potential solution could be to add one linear layer to the phoneme stream that takes the one hot encoded phonemes of any given language as input and outputs

the required size of the English one hot encoded phonemes.

5.6 Clusters

We have seen that the clusters generated by the ConvGRU model were of higher quality than those generated by the 3D-CNN. Firstly, we saw how the 3D-CNN model produced a duplicate representative pair. Two different clusters having the same representative gives a big warning that something is going wrong. Ideally we want all duplicates of a unique word to be in the same cluster as their articulation pattern is supposed to be the same. On further informal inspection of the clusters created by the ConvGRU, the set of representatives includes shorter words, longer words and does not contain many words that either start or end the same. Furthermore, we can see in Figure 8b that the representative words gradually increase in length when we look from the upper left to the bottom right. Similar patterns of gradual increase in word length across the cluster space can be observed for the fine-tuned models.

On more formal inspection, the SD scores further show the difficulties the 3D-CNN model had with effectively clustering duplicates together. The ALD scores were worse for the 3D-CNN compared to the ConvGRU and the ACCD was equal for the two architectures. However, where the ConvGRU showed a significant clustering of phonemic Levenshtein information, the 3D-CNN failed to produce a significant result. Therefore, by informal and formal inspection of the cluster spaces of the two architectures, we can conclude that the clusters generated by the ConvGRU embeddings were of higher quality than those generated by the 3D-CNN embeddings.

However, as mentioned before, the clustering of data can be done in many different ways, even by the same algorithm. It is likely that setting the number of clusters to 20 did not lead to the most optimal clustering. We have set this number to 20 for the pragmatic reason of providing as much degrees of freedom for the user of a BCI application while staying manageable for the application engineers. Brief experimentation with a range of cluster counts between 2 and 100 resulted in an optimal clustering when the cluster count was 2 and gradually decreased in quality when the number of clusters increased. Though 2 words does not provide a pragmatic solution for BCI applications, this experiment does suggest there is a trade-off to be made between the amount of words to include and the overall reliability of the system. With less words, the system will have high reliability but low expressivity, and with more words the system will have higher expressivity but lower reliability.

Jaw	Crooked	And	Irate	Annoying
Sought	Teaspoons	So	All	Exciting
Triumphant	Original	Anecdotal	Items	Confirm
Novel	Frustration	Spray	Orange	Bagpipes

Table 6: 20 most distinct articulation patterns

Since the models were trained on a dataset consisting of only 2800 unique words, it is likely that there are more distinct articulation patterns to be found in the words outside of this dataset. The English language consists of over 1 million words, meaning that many words that could have been very distinct in their articulation pattern were never considered in this research. Moreover, as mentioned in section 4.1, the quality of the data itself could also be improved on multiple aspects (frame rate, sound quality, transcription), which would lead to higher quality embedding spaces making the clustering more reliable. Therefore, this research wants to recommend the use of the 20 cluster representatives generated by the ConvGRU approach (shown in Table 6) as the 20 most discriminable words for direct word encoding BCI applications. Additionally, one of the benefits of the ConvGRU approach is that the number of clusters is a parameter and can therefore be changed to fit what the researchers need. For the research of the earlier mentioned Moses et al. (2021) for example, instead of using the 50 words they chose because they are common in the English language, we would recommend them to use our ConvGRU approach with 50 clusters to get 50 words that should be more distinct from each other and therefore more reliably decoded. However, in such cases, we do recommend to use the ConvGRU approach on data consisting of only manually transcribed single word spoken rtMRI videos. This data should be better suitable and is therefore expected to generate better word embeddings and consequently better clusters/representatives.

6 Conclusion

Following our results, a convolutional recurrent approach outperforms a pure convolutional one when generating word embeddings from rtMRI videos. Furthermore, these embeddings get closer to EMA observations when we provide the model with phonemic content information. The results generalized well over multiple participants suggesting that the results found on one healthy participant also count for other healthy people, and therefore potentially for people with LIS. We also found that our autoencoders could easily be fine-tuned to new participants allowing them to be a valuable addition to the articulatory research field as generating word embeddings for a new participant does not require large training set sizes. With our best performing autoencoder architecture we determined the 20 words that should be the most distinct in their articulation pattern, which following the literature should be the most distinct in their neural patterns. Therefore, these 20 words provide the set of most reliably decoded words in direct word encoding BCI applications.

References

Akgul, Y. S., Kambhamettu, C., and Stone, M. (1998). Extraction and tracking of the tongue surface from ultrasound image sequences. In *Proceedings. 1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No. 98CB36231)*, pages 298–303. IEEE.

Al-Hammadi, M., Muhammad, G., Abdul, W., Alsulaiman, M., and Hossain, M. S. (2019). Hand gesture recognition using 3d-cnn model. *IEEE Consumer Electronics Magazine*, 9(1):95–101.

Albrecht, G. L. and Devlieger, P. J. (1999). The disability paradox: high quality of life against all odds. *Social science & medicine*, 48(8):977–988.

Amiriparian, S., Freitag, M., Cummins, N., and Schuller, B. (2017). Sequence to sequence autoencoders for unsupervised representation learning from audio. In *DCASE*, pages 17–21.

Assent, I. (2012). Clustering high dimensional data. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(4):340–350.

Ballas, N., Yao, L., Pal, C., and Courville, A. (2015). Delving deeper into convolutional networks for learning video representations. *arXiv preprint arXiv:1511.06432*.

Bank, D., Koenigstein, N., and Giryes, R. (2020). Autoencoders.

Bauby, J.-D. (2008). *The diving bell and the butterfly*. Vintage.

Bauer, G., Gerstenbrand, F., and Rumpl, E. (1979). Varieties of the Locked-in Syndrome. Technical report.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

Bouchard, K. E., Mesgarani, N., Johnson, K., and Chang, E. F. (2013). Functional organization of human sensorimotor cortex for speech articulation. *Nature*, 495(7441):327–332.

Breuel, T. M. (2015). Benchmarking of lstm networks. *arXiv preprint arXiv:1508.02774*.

Bruno, M.-A. L., Bernheim, J. L., Ledoux, D., Dé, F., Pellas, R., Demertzi, A., and Laureys, S. (2011). A survey on self-assessed well-being in a cohort of chronic locked-in syndrome patients: happy majority, miserable minority.

Budden, D., Matveev, A., Santurkar, S., Chaudhuri, S. R., and Shavit, N. (2017). Deep tensor convolution on multicores. In *International Conference on Machine Learning*, pages 615–624. PMLR.

Chartier, J., Anumanchipalli, G. K., Johnson, K., and Chang, E. F. (2018). Encoding of Articulatory Kinematic Trajectories in Human Speech Sensorimotor Cortex. *Neuron*, 98(5):1042–1054.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Chong, Y. S. and Tay, Y. H. (2017). Abnormal Event Detection in Videos using Spatiotemporal Autoencoder.

Csapó, T. G. (2020). Speaker dependent articulatory-to-acoustic mapping using real-time MRI of the vocal tract.

Dastider, A. G., Sadik, F., and Fattah, S. A. (2021). An integrated autoencoder-based hybrid CNN-LSTM model for COVID-19 severity prediction from lung ultrasound. *Computers in Biology and Medicine*, 132.

Doble, J. E., Haig, A. J., Anderson, . C., and Katz, R. (2003). Impairment, Activity, Participation, Life Satisfaction, and Survival in Persons With Locked-In Syndrome for Over a Decade Follow-Up on a Previously Reported Cohort. Technical Report 5.

Dumoulin, V. and Visin, F. (2016). A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.

Dwarampudi, M. and Reddy, N. (2019). Effects of padding on lstms and cnns. *arXiv preprint arXiv:1903.07288*.

Fukushima, K. and Miyake, S. (1982). Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer.

Gilja, V., Pandarinath, C., Blabe, C. H., Nuyujukian, P., Simeral, J. D., Sarma, A. A., Sorice, B. L., Perge, J. A., Jarosiewicz, B., Hochberg, L. R., Shenoy, K. V., and Henderson, J. M. (2015). Clinical translation of a high-performance neural prosthesis. *Nature Medicine*, 21(10):1142–1145.

Haugen, T. B., Hicks, S. A., Andersen, J. M., Witczak, O., Hammer, H. L., Borgli, R., Halvorsen, P., and Riegler, M. (2019). VISEM: A multimodal video dataset of human spermatozoa. In *Proceedings of the 10th ACM Multimedia Systems Conference, MMSys 2019*, pages 261–266. Association for Computing Machinery, Inc.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Honda, K. (1983). Relationship between pitch control and vowel articulation. *Haskins Laboratories Status Report on Speech Research, SR*, 73:269–282.

Hoult, D. I. and Bhakar, B. (1997). Nmr signal reception: Virtual photons and coherent spontaneous emission. *Concepts in Magnetic Resonance: An Educational Journal*, 9(5):277–297.

Hubel, D. H. and Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106.

Hubel, D. H. and Wiesel, T. N. (1968). Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243.

Ji, S., Xu, W., Yang, M., and Yu, K. (2012). 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231.

Ji, S., Xu, W., Yang, M., and Yu, K. (2013). 3D Convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231.

Katz, W. F., Bharadwaj, S. V., and Stettler, M. P. (2006). Influences of electromagnetic articulography sensors on speech produced by healthy adults and individuals with aphasia and apraxia.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization.

Kübler, A., Winter, S., Ludolph, A. C., Hautzinger, M., and Birbaumer, N. (2005). Severity of depressive symptoms and quality of life in patients with amyotrophic lateral sclerosis. *Neurorehabilitation and neural repair*, 19(3):182–193.

Laureys, S., Pellas, F., Van Eeckhout, P., Ghorbel, S., Schnakers, C., Perrin, F., Berre, J., Faymonville, M.-E., Pantke, K.-H., Damas, F., et al. (2005). The locked-in syndrome: what is it like to be conscious but paralyzed and voiceless? *Progress in brain research*, 150:495–611.

León-Carrión, J., Van Eeckhout, P., and Domínguez-Morales, M. D. R. (2002). The locked-in syndrome: A syndrome looking for a therapy.

Levenshtein, V. I. et al. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710. Soviet Union.

Lopez-del Rio, A., Martin, M., Perera-Lluna, A., and Saidi, R. (2020). Effect of sequence padding on the performance of deep learning models in archaeal protein functional prediction. *Scientific reports*, 10(1):1–14.

Lulé, D., Zickler, C., Häcker, S., Bruno, M. A., Demertzi, A., Pellas, F., Laureys, S., and Kübler, A. (2009). Life can be worth living in locked-in syndrome.

Meng, Q., Catchpoole, D., Skillicorn, D., and Kennedy, P. J. (2018). Relational Autoencoder for Feature Extraction.

Moses, D. A., Metzger, S. L., Liu, J. R., Anumanchipalli, G. K., Makin, J. G., Sun, P. F., Chartier, J., Dougherty, M. E., Liu, P. M., Abrams, G. M., et al. (2021). Neuroprosthesis for decoding speech in a paralyzed person with anarthria. *New England Journal of Medicine*, 385(3):217–227.

Mugler, E. M., Tate, M. C., Livescu, K., Templer, J. W., Goldrick, M. A., and Slutzky, M. W. (2018). Differential representation of articulatory gestures and phonemes in precentral and inferior frontal gyri. *Journal of Neuroscience*, 38(46):9803–9813.

Narayanan, S., Toutios, A., Ramanarayanan, V., Lammert, A., Kim, J., Lee, S., Nayak, K., Kim, Y.-C., Zhu, Y., Goldstein, L., Byrd, D., Bresch, E., Ghosh, P., Katsamanis, A., and Proctor, M. (2014). Real-time magnetic resonance imaging and electromagnetic articulography database for speech production research (TC). *The Journal of the Acoustical Society of America*, 136(3):1307–1311.

Nuyujukian, P., Albites Sanabria, J., Saab, J., Pandarinath, C., Jarosiewicz, B., Blabe, C. H., Franco, B., Mernoff, S. T., Eskandar, E. N., Simeral, J. D., Hochberg, L. R., Shenoy, K. V., and Henderson, J. M. (2018). Cortical control of a tablet computer by people with paralysis. *PLoS ONE*, 13(11).

Olivia Gosseries, Marie-Aurélie Bruno, Audrey Vanhaudenhuyse, Steven Laureys, and Caroline Schnakers (2009). Consciousness in the Locked-in Syndrome. Technical report.

Pandarinath, C., Gilja, V., Blabe, C. H., Nuyujukian, P., Sarma, A. A., Sorice, B. L., Eskandar, E. N., Hochberg, L. R., Henderson, J. M., and Shenoy, K. V. (2015). Neural population dynamics in human motor cortex during movements in people with als. *eLife*, 4:e07436.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Patterson, J. R. and Grabois, M. (1986). Locked-In Syndrome: A Review of 139 Cases. Technical report.

Rabkin, J. G., Wagner, G. J., and Del Bene, M. (2000). Resilience and distress among amyotrophic lateral sclerosis patients and caregivers. *Psychosomatic medicine*, 62(2):271–279.

Rebernik, T., Jacobi, J., Jonkers, R., Noiray, A., and Wieling, M. (2021). A review of data collection practices using electromagnetic articulography. *Laboratory Phonology*, 12(1).

Rousseau, M. C., Baumstarck, K., Alessandrini, M., Blandin, V., Billette De Villemeur, T., and Auquier, P. (2015). Quality of life in patients with locked-in syndrome: Evolution over a 6-year period. *Orphanet Journal of Rare Diseases*, 10(1).

Saito, M., Tomaschek, F., and Baayen, R. H. (2021). An ultrasound study of frequency and co-articulation.

Schönle, P. W., Gräbe, K., Wenig, P., Höhne, J., Schrader, J., and Conrad, B. (1987). Electromagnetic articulography: Use of alternating magnetic fields for tracking movements of multiple points inside and outside the vocal tract. *Brain and Language*, 31(1):26–35.

Schultz, T. and Wand, M. (2010). Modeling coarticulation in emg-based continuous speech recognition. *Speech Communication*, 52(4):341–353.

Shi, X., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K., Woo, W.-C., and Kong Observatory, H. (2015). Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. Technical report.

Smith, E. and Delargy, M. (2005). Locked-in syndrome.

Srivastava, N., Mansimov, E., and Salakhutdinov, R. (2015). Unsupervised Learning of Video Representations using LSTMs.

Steinley, D. (2006). K-means clustering: a half-century synthesis. *British Journal of Mathematical and Statistical Psychology*, 59(1):1–34.

Tran, D., Bourdev, L., Fergus, R., Torresani, L., and Paluri, M. (2015). Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497.

Vansteensel, M. J., Pels, E. G., Bleichner, M. G., Branco, M. P., Denison, T., Freudenburg, Z. V., Gosselaar, P., Leinders, S., Ottens, T. H., Van Den Boom, M. A., Van Rijen, P. C., Aarnoutse, E. J., and Ramsey, N. F. (2016). Fully Implanted Brain–Computer Interface in a Locked-In Patient with ALS. *New England Journal of Medicine*, 375(21):2060–2066.

Vidal, F. (2020). Phenomenology of the Locked-In Syndrome: an Overview and Some Suggestions. *Neuroethics*, 13(2):119–143.

Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2014). Show and tell: A neural image caption generator. *CoRR*, abs/1411.4555.

Wagner, W. (2010). Steven bird, ewan klein and edward loper: Natural language processing with python, analyzing text with the natural language toolkit. *Language Resources and Evaluation*, 44(4):421–424.

Wattenberg, M., Viégas, F., and Johnson, I. (2016). How to use t-sne effectively. *Distill*, 1(10):e2.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83.

Willett, F. R., Avansino, D. T., Hochberg, L. R., Henderson, J. M., and Shenoy, K. V. (2021). High-performance brain-to-text communication via handwriting HHS Public Access. *Nature*, 593(7858):249–254.

Wilson, I. (2014). Using ultrasound for teaching and researching articulation. *Acoustical Science and Technology*, 35(6):285–289.

Wolpaw, J. R. (2007). Brain-computer interfaces (bcis) for communication and control. In *Proceedings of the 9th international ACM SIGACCESS conference on Computers and accessibility*, pages 1–2.

Yu, Y., Shandiz, A. H., and Tóth, L. (2021). Reconstructing Speech from Real-Time Articulatory MRI Using Neural Vocoders.

Yumang, A. N., Villaverde, J. F., Padilla, D. A., and Gatdula, M. M. V. (2020). Environmental control system for locked-in syndrome patients using eye tracker. In *Proceedings of the 2020 10th International Conference on Biomedical Engineering and Technology*, pages 234–239.

Zhao, X., Wei, H., Wang, H., Zhu, T., and Zhang, K. (2019). 3D-CNN-based feature extraction of ground-based cloud images for direct normal irradiance prediction. *Solar Energy*, 181:510–518.