# TEICUN: A Transformer Encoder Infused Convolutional UNet for EEG Motion Artifact Removal

Alexander Suddaby

**Abstract**

The World Stroke Organisation (WSO) states that 1 in 4 people over the age of 25 will have a stroke in their lifetime [1]. Intervention is required to help rehabilitate stroke survivors and help them regain the maximum quality of life. Current rehabilitation techniques focus physical exercise and assessment but often do not monitor the brain directly. This would be useful in the treating of brain injury to be able to give more direct feedback and inform decision making.

Currently, methods of assessing brain activity are limited by cost and their ability to work in motion. EEG is one of the cheapest methods for reading brain activity but suffers from artifacts caused by motion. This research aimed to address this issue by removing motion contamination from EEG signals. To do this, existing processing pipelines were assessed. These assessments covered both traditional and machine learning algorithms with the aim of identifying areas for innovation.

Using PyTorch, and other available libraries, this research evaluated existing preprocessing methods and proposed new methods with a new machine learning architecture. The resulting architecture is a convolutional UNet with bottleneck replaced with a transformer encoder. The proposed preprocessing steps achieved a significant temporal correlation of 86.883% and a 26.896 dB improvement in SNR. These improvements highlight the method's ability to preserve signal integrity while removing motion artifacts. Notably, the network accomplished this performance with around 173,000 fewer parameters than alternative methods, demonstrating state-of-the-art performance. This reduction in parameters reduces computational costs making it more suitable for real-time application in resource-constrained environments such as a clinical setting.

However, comparison with other techniques is difficult because of differences in preprocessing techniques. The nuances of this are also discussed in detail.

# TEICUN: A Transformer Encoder Infused Convolutional UNet for EEG Motion Artifact Removal

Alexander Suddaby

Master of Science

Department of Computer Science

Durham University

2024

# Contents

# List of Figures

# List of Tables

# Acronyms

**BCE** Binary Cross-Entropy. 56, 57

**BCI** Brain Computer Interface. 10

**BEI** Brain Engagement Index. 15

**CEEMDAN+AF** Complete Ensemble Empirical Mode Decomposition with Adaptive Noise. 22

**CNN** Convolutional Neural Network. 21

**CQC** Care Quality Commission. 10

**CSV** Comma Separated Value. 39

**DDR4** Double Data Rate 4. 35

**DWT** Discrete Wavelet Transform. 18

**ECG** Electrocardiogram. 22

**EEG** Electroencephalography. 1, 10–19, 21–26, 29, 32–37, 39, 40, 42, 46, 49, 54, 58, 59, 86, 87, 93, 94, 96, 97

**EEMD-CCA** Ensemble Empirical Mode Decomposition - Canonical Correlation Analysis. 17

**EEMD-ICA** Ensemble Empirical Mode Decomposition - Independent Component Analysis. 17

**EPSP** Excitatory Postsynaptic Potential. 13

**FID** Fréchet Inception Distance. 24

**FIR** Finite Impulse Response. 15

**fMRI** functional Magnetic Resonance Imaging. 10

**fNIRS** functional Near-Infrared Spectroscopy. 10

**FPN** Feature Pyramid Network. 3, 4, 19, 21, 27, 47–49, 72, 73

**GAN** Generative Adversarial Network. 3, 25–31

**GB** Gigabytes. 35

**GELU** Gaussian Error Linear Unit. 50, 51

**IIR** Infinite Impulse Response. 15, 40, 41, 43

**IPSP** Inhibitory Postsynaptic Potential. 13

**IQ** Information Quality. 4, 23

**KID** Kernel Inception Distance. 24

**LSTM** Long Short Term Memory. 23, 24

**MAE** Mean Absolute Error. 46, 57

**ML** Machine Learning. 35, 45

**MRP** Multi-resolution Pooling. 21

**MSE** Mean Squared Error. 46, 57, 66

**MTV** Multi-resolution Total Variation. 18

**MWTV** Multi-resolution Weighted Total Variation. 18

**NHS** National Health Service. 10

**ONN** Operational Neural Network. 21, 51, 52

**PCC** Pearson Correlation Coefficient. 3–5, 36–38, 40–42, 59, 61–64, 66–80, 82–87, 90–92, 95

**PSP** Postsynaptic Potential. 13

**RAM** Random Access Memory. 35

**ReLU** Rectified Linear Unit. 19, 20, 46, 50

**RMS** Root-Mean-Square. 57

**RNN** Recursive Neural Network. 23

**RVFLN** Random Vector Functional Link Network. 22

**Self-ONN** Self-Organised Operational Neural Network. 1–4, 21, 51–54, 56, 74–80

**SGD** Stochastic Gradient Descent. 31

**SLFN** Single Hidden-layer Feed-forward Network. 22

**SNR** Signal-to-Noise Ratio. 1, 4, 16–18, 22, 23, 87, 88, 90–92, 97

**SSA** Singular Spectrum Analysis. 17, 18

**TV** Total Variation. 18

**ViT** Vision Transformer. 24, 49

**VR** Virtual Reality. 10

**VRAM** Video Random Access Memory. 35

**WGAN** Wasserstein GAN. 28

**WSO** World Stroke Organization. 10

**WTV** Weighted Total Variation. 18

## Statement of Copyright

# 1 Introduction

According to the World Stroke Organization (WSO), there are 12.2 million new people affected by strokes each year, with 1 in 4 people over the age of 25 experiencing a stroke in their lifetime [1]. The likelihood of developing a neurological disorder increases with age with women being more likely to suffer than men [2]. With increasing demand and pressures on health care professionals having detrimental effects on the well-being of staff in the National Health Service (NHS) England [3], intervention and innovation is required to support stretched workforces. In the same report from the Care Quality Commission (CQC), healthcare professionals have stated that inadequate support is effecting the quality of care they are able to provide. In addition, non-adherence to rehabilitation programmes ranges from 50% to as high as 70% [4], [5] is resulting in patients receiving sub-optimal rehabilitation.

A better understanding of this problem could be found using a Brain Computer Interface (BCI) which provide a quantitative measure of brain activity. Using a BCI, it is possible to measure engagement and predict the emotional response of a patient by evaluating the recorded signals. Being able to identify exercises and activities which are most engaging to a patient has the potential to provide great benefits in terms of planning rehabilitation programmes. With this information, it would be possible to tailor rehabilitation programmes on an individual level, in theory increasing adherence and overall outcome. Combining this with emerging technologies such as Virtual Reality (VR) could create a more engaging environment which can dynamically adapt to patient needs and interests. Recommending particular exercises based on individual characteristics has the potential to increase adherence to rehabilitation programmes.

There are many techniques currently in place for reading brain activity: functional Near-Infrared Spectroscopy (fNIRS), functional Magnetic Resonance Imaging (fMRI) and Electroencephalography (EEG) are examples of some commonly used techniques. Whilst all of these methods are useful from gathering brain data, fNIRS and fMRI are both limited by portability and cost. EEG on the other hand, offers a way of measuring brain activity at a relatively low cost.

EEG being a portable and relatively cost-effective technique means it has the potential to be useful for measuring people completing activities. For example, it could be useful for monitoring patients participating in rehabilitation exercises or completing other physical activities. It has the potential to be able to monitor progress, engagement and predict emotions towards the current activity.

However, EEG is limited by multiple issues. Firstly, it is only able to record

surface-level brain activity. Whilst this is not a particular issue due to the structure of the brain, it means that information can be diluted and obfuscated. The second is that EEG uses very sensitive electrodes to read brain signals. As a result, motion can cause large artifacts from loose wires adding noise to the signal. This limitation means that participants being studied and recorded are required to be very still during recording sessions.

EEG is also extremely sensitive to external stimuli. Due to the strength of the signals picked up, EEG requires amplification which is affected by environmental noise such as electromagnetic interference from power lines or contamination from motion of a subject. While environmental noise can largely be filtered out through the use of a notch filter at the appropriate frequency for the region (50 Hz / 60Hz), motion contamination remains an issue. Patients wearing the headset will be required to move to complete their exercises. As such, the motion contamination needs to be reduced or removed to allow for an accurate assessment.

The usefulness of EEG whilst users move has the potential to contribute to patient care. Artificial Intelligence (AI) provides a potential way to tackle this issue by using various techniques to create an intelligent solution to the problem. This research aims to tackle the issue of motion contamination in EEG signals by using modern deep learning techniques.

# 2 Contributions

This research offers multiple contributions to the task of removing motion contamination from EEG signals. A light weight neural network containing 1,779,953 parameters was produced using a technique using a convolutional UNet with the bottleneck replaced with a transformer encoder. This network contains fewer parameters than the best performing direct comparison [6] (containing around 1.953 million parameters). It also achieves an improvement in temporal correlation over the network of around 1.08% (to 89.08%). Both of these factors highlight the contribution of the network architecture to the challenge of motion contamination removal from EEG signals.

In addition to the network architecture improvements, the proposed data preprocessing and windowing technique also contributes to the improvement in results. This builds on existing methods by applying techniques which can be applied in real-time. This is done by only apply preprocessing to signals which have already

been downsampled and windowed to the correct size for processing by the network. This allows for signals to be analysed during exercises and trials allowing for instant feedback.

Finally, the proposed windowing method increases the focus on severe motion contamination for network training purposes. This was accomplished by dynamically changing the window overlap based on the temporal correlation between the ground truth and motion contaminated signal. By increasing overlap the worse the temporal correlation, greater focus was given to training the network to remove motion contamination rather than just reproducing the input signal. 43.16% of the resulting dataset are segments with a temporal correlation, between the ground truth and motion contaminated signals, of below 50% prior to preprocessing. This is in contrast to 27.7% using the method from [6] displaying an improvement on the focus on poorly correlated segments.

Overall, this work contributes to the removal of motion contamination from EEG signals by designing and training a network with fewer parameters than the closest direct competition. This is done by introducing a lightweight transformer-encoder-based UNet architecture. Reducing parameters while maintaining state-of-the-art performance.

In addition, it iterates and improves existing pre-processing techniques for the training datasets in order to find improvements in the motion contamination removal process. These steps allow for a more computationally efficient processing pipeline which reduces overall costs. In turn, this cost reduction is a step towards the implementation of such systems in a practical resource-constrained setting. This includes clinical settings where such systems could allow for greater understanding of patient needs and requirements, enabling tailored rehabilitation.

# 3 Background

## 3.1 Brainwaves

To study the brain effectively, it is first important to understand brainwaves, which are categorised by frequency and are indicative of different cognitive states [7]. Delta waves, ranging between 0.1 Hz and 4 Hz, are commonly associated with deep sleep, important for physical and mental restoration. Theta waves span 4 Hz to 8 Hz and are linked to dreaming and visualisation, often observed during light sleep. Alpha waves, occurring at frequencies of 8 Hz to 13 Hz, are correlated with creativity and

relaxation.

In contrast, beta waves are associated with heightened alertness, critical thinking and concentration. Spanning from 13 Hz to 30 Hz, beta waves are an indicator of an individuals engagement with the external environment. Finally, gamma waves are waves with a frequency between 30 Hz and 100 Hz. Gamma waves are associated with cognitive processing, peak experiences and synchronisation of neural activity, suggesting a state of heightened perception. Understanding these brainwaves is of the utmost importance when studying the brain.

Brainwaves like these can be measured due to the electrical properties of the brain which is made up of vast networks of neurons. Neurons communicate using chemical and electrical signals. When a signal is sent from one neuron to another, a small electrical change occurs in the receiving neuron, known as a Postsynaptic Potential (PSP). PSPs can either make the receiving neuron more likely to send a signal to the next neuron (Excitatory Postsynaptic Potential (EPSP)) or less likely (Inhibitory Postsynaptic Potential (IPSP)) [8]. Although the electrical potential of a single neuron firing is too small to be detected from the surface of the scalp, it is possible to measure the activity over a larger area from the cerebral cortex, the outer layer of the brain. The frequency of the detected activity can then be interpreted and analysed for an insight into how an individual's brain is functioning.

## 3.2 EEG

EEG is a common technique used for measuring brain activity. It is a non-invasive technique in which a subject has multiple electrodes placed on their head. These electrodes are used to measure voltage differences across the surface of the brain. These signals can be differentiated and interpreted using a number of different factors, such as location and frequency power.

Commonly, an EEG reading is taken using a cap with electrodes placed in the 10 - 20 system. The 10 - 20 system is an internationally recognised, standardised way of placing electrodes on the head of a subject. The system is named after the placement of the electrodes which are placed either 10% or 20% from the centre of the scalp. Determining the centre of the scalp is accomplished by measuring from the nasion (the join between the forehead and the nose) and the inion (the base of the skull at the back of the head) and finding the centre point, see figure 1. The distance between the preauricular points on either side of the head, also in figure 1, are also measured and the centre point marked. The place where these 2 measurements cross over is the

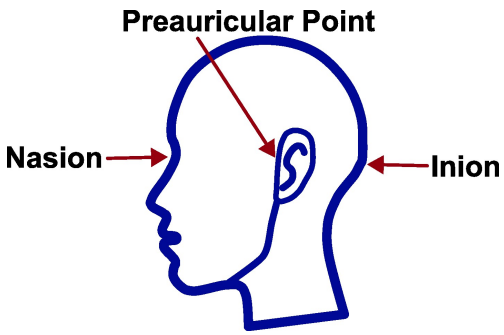



Figure 1: Inion, Nasion and Preauricular points

Figure 2: Active Electrode available from G.Tec [9]

desired location of the central electrode. The cap is placed on the subject's head using this as a reference point. Once placed on the subject's head in the correct position, it should be ensured that all electrodes are in contact with the scalp and that the headset is tightly secured.

There are 2 common types of electrodes used for EEG. The first are passive electrodes. Passive electrodes do not amplify the detected signals and require the use of a conductive gel. The setup process for passive electrodes involves injecting this gel through a hole in the centre of the electrodes to ensure a strong signal is being detected. This results in high accuracy signals but also leads to longer clean up and preparation times. Passive electrodes are used in the majority of EEG studies for the benefit of increased accuracy. Active electrodes are the second commonly used type of electrode. The difference being that active electrodes amplify the signals at the source. A benefit of this is that conductive gel is not required. Whilst this can result in poorer signal quality, it leads to shorter preparation and clean up times as well as being generally more comfortable to wear.

Once data has been collected, signals may require post-processing and evaluating depending on the requirements. As EEG is prone to a number of different artifacts, this is an essential step to clean the data and make it usable.

EEG data is prone to baseline drift. This is when the signal gradually increases or decreases overtime and can be caused by a number of factors including electrodes slipping from their initial positions and changes in temperature. Baseline drift should be removed when working with EEG data to improve the quality of recordings and interpretations. For this purpose, the trend is commonly removed from EEG data. There are multiple approaches to this including: linear detrending, high-pass filtering and moving average [10]. Linear detrending aims to fit a polynomial to the general trend of the data which is then removed.

Detrending is a common technique which involves fitting a polynomial to the general trend of the EEG data. This polynomial is then subtracted from the data. Baseline drift can be accounted for by detrending methods and it has limited impact on the data.

Many techniques in EEG post-processing involve using Infinite Impulse Response (IIR) filters or Finite Impulse Response (FIR) filters [11]. Both filters are useful for blocking and removing frequencies from a signal. IRR filters are recursive and computationally efficient, making them useful for real-time applications. FIR filters are non-recursive which makes them more stable than IRR filters but more computationally expensive. Either of these techniques can be combined to create a number of different operations such as band-stop, band-pass, high-pass and low-pass filters.

Band-stop filters remove frequencies from a signal [11]. EEG is prone to environmental interference such as noise from power-lines. As such, it is useful to be able to remove a particular range of frequencies from the signal (normally 50Hz or 60Hz depending on which region the recordings are taken in). This is sometimes also referred to as a notch filter.

Band-pass filters extract a specified range of frequencies from the signal [12]. Frequencies of brain activity are associated with different mental states. Being able to isolate these states is useful for a number of different assessment criteria, for example the Brain Engagement Index (BEI) which is used to quantify the engagement of participants [13]–[15]. Band-pass filters are useful for extracting a given frequency range from the data to use for such calculations. The equation of the engagement index can be found below.

$$Engagement = \frac{\beta}{\alpha + \theta}$$

High-pass and low-pass filters filter out frequencies above or below a certain frequency respectively. This is useful for removing low frequencies which impact useful

data or removing high frequencies which are out of the range of interest. For example, gamma brainwaves are the highest frequency brainwaves typically researched but there is negligible data to be found above 100 Hz so it could be useful to remove frequencies above this level to simplify the data.

More recent techniques aim to improve traditional techniques by being able to detrend and inpaint data [16] . While this technique claims to provide better quality detrending of EEG data, it relies on the use of multiple data channels making it unusable on the available dataset.

EEG headsets require careful setup and constant analysis during trials and experiments to ensure any recorded data is reliable. Knowledge of how EEG equipment should be setup and monitored is an important part of the process. In addition, knowledge of basic preprocessing techniques, such as bandpass filters, is essential for the assessment of signal quality and for the extraction of useful data.

# 4 Literature Review

## 4.1 Overview

Removing motion artifact contamination from EEG signals has been a focus of research for over a decade. Previously, there was no framework for assessing the accuracy of improvements made other than assessing the improvement of Signal-to-Noise Ratio (SNR).

In 2012, a dataset was created to allow motion contamination removal methods to be evaluated [17]. Creating a matched dataset for motion artifact contamination is incredibly difficult, since it is not possible to record someone who is stationary and moving at the same time. To work around this, the dataset was created by having stationary participants wear 2 electrodes within 30 mm of each other. This ensured that the readings are highly correlated. One of the electrodes was periodically tapped for the duration of the recording, creating artificial motion artifact contamination on one of the electrodes. To supplement this, an accelerometer was attached to each electrode, allowing the difference in motion to be used for motion artifact removal.

The "Motion Artifact Contaminated fNIRS and EEG Data" dataset is a useful tool for evaluating methods of removing motion artifact contamination from EEG signals and is the best method currently available for assessing how well algorithms remove the motion contamination. It is currently the only well known and freely available matched dataset for EEG motion contamination removal. However, it is limited by

size. Only 23 participants were recorded for around 9 minutes each. This lack of diversity in the dataset could lead to results which do not generalise well with a larger sample. Ideally, a larger dataset would be used for verification.

It is also limited by the use of artificial motion contamination. Although it would be impossible to create a matched dataset any other way, without the use of simulation, it means that other potential indicators of impending motion, such as an increase in a particular frequency of brain activity, cannot be taken into account. In addition, it also removes the possibility of working with signals from multiple electrodes. Any motion made by a participant wearing an EEG headset should have a similar impact on all EEG channels. Assessing multiple channels at once would allow for more robust motion contamination removal. Nevertheless, the dataset provides a quantifiable way of testing motion contamination removal methods.

## 4.2 Classical Approaches

The "Motion Artifact Contaminated fNIRS and EEG Data" dataset was initially used to assess multiple methods removing motion artifacts [17]. Using the collected dataset, the accuracy of adaptive filters, Kalman filters and Ensemble Empirical Mode Decomposition - Independent Component Analysis (EEMD-ICA) was assessed. Kalman filters demonstrated the best improvements in temporal correlation (83.13%) and SNR (9.7dB), followed by EEMD-ICA (76.5%, 8.9dB) and finally adaptive filters (37.6%, 5.1dB). Improvement in this area since the creation of this dataset has been easier to measure and assess.

The use of classical algorithms has since been used to improve the performance of motion artifact removal. One such approach used Singular Spectrum Analysis (SSA) [18]. In this approach, SSA was suggested for a number of reasons including: low computational complexity compared to other methods; its adaptability to single-channel data and its effectiveness in isolating low-frequency signal artifacts. This approach was chosen because motion contamination often appears as low-frequency artifacts and the available dataset, "Motion Artifact Contaminated fNIRS and EEG Data", is only single-channel data. This aimed to improve upon Ensemble Empirical Mode Decomposition - Canonical Correlation Analysis (EEMD-CCA), originally suggested in 2006 [19] - predating the creation of the dataset.

One detail of the paper is the exclusion of participant data. Participants 12 and 15 were removed from the dataset due to poor correlation even over the clean epochs. Following visual inspection, the research claims that no brain activity is present in

the signals recorded for these participants. After this, the results were compared to EEMD-CCA. The research found SSA to have a 0.91507 dB improvement of SNR ($\Delta$SNR ) over EEMD-CCA as well as an 11.388% improvement in temporal correlation. In addition, this comes with significantly higher performance. The paper claims a reduction of computational complexity of around 5 times.

Multi-resolution Total Variation (MTV) and Multi-resolution Weighted Total Variation (MWTV) denoising are methods which have also been used to remove motion contamination from EEG signals [20]. These methods use a Discrete Wavelet Transform (DWT) for the extraction of sub-band signals from the motion contaminated signal. Either Total Variation (TV) or Weighted Total Variation (WTV) are then applied over the DWT approximation. The difference between the approximated sub-band signal and output of the TV or WTV filter is used to create a filtered approximate signal. This approach did not leave out any of the recordings from the dataset and claimed around a 29.12 dB improvement in $\Delta SNR$, from the original dataset, as well as a temporal correlation of around 68.56% using MTV denoising. Using MWTV denoising, a $\Delta SNR$ improvement of 29.29 dB was found with a temporal correlation of 67.37%.

MTV and MWTV offered significant improvement of $\Delta SNR$ over the Kalman filter which was the best result from [17], an improvement of around 19.6 dB. However, it fails to reach the temporal correlation achieved by the Kalman filter. Here it is arguable that the SNR is a more significant measure of accuracy as it represents the true transformation from the motion contaminated domain to the ground truth domain. Although SNR can be useful for assessing signal quality, if it improves beyond that of the ground truth signal, it is likely that solely optimising this metric would lead to removing important information from the signal.

This research is separated by over half a decade showing that traditional methods are slow to advance and limited progress has been made. While traditional methods can improve the impact of motion artifacts in EEG signals, the drawbacks become obvious when looking at existing research.

Traditional algorithms are limited by relying on a predefined hypothesis making them more likely to overlook unexpected predictor values [21]. It is difficult for anyone to look at a large dataset and create a mathematical model to fit a problem that can take all the appropriate factors into account. For these reasons, machine learning seems a much better fit for such a complex task. Recent research has looked to machine learning for more accurate algorithms which can fit the provided dataset.

## 4.3 Machine Learning Approaches

More recently, machine learning has been used to improve the performance of motion artifact removal from EEG signals. In 2023, [22], used a UNet inspired network, along with the "Motion Artifact Contaminated fNIRS and EEG Data" dataset, to tackle this problem. Following this, they also experimented with a modified Feature Pyramid Network (FPN) [6] and claimed significant improvement over existing traditional techniques. These approaches both work by augmenting the dataset and segmenting it into 4 second windows, using differing methods.

[22], as opposed to [6], makes use of some preprocessing applied to both the ground truth and motion contaminated signals. This has the benefit of simplifying the task for the neural network as it can remove artifacts, in tried and tested ways, which are not being targeted by the network. Consequently, this allows the network to become more focused a specialised at a particular task. However, between 2015 and 2020, 21.28% of hybrid Deep Learning post-processing of EEG data did not apply a preprocessing step [23]. This suggests that there is scope to further improve existing methods through the introduction of preprocessing.

Both [22] and [6] make use of encoder-decoder architectures. An example of this is a UNet. The work of [22] used an adapted UNet for EEG motion artifact removal. A UNet is a convolutional neural network that was originally designed for biomedical image segmentation [24]. Biomedical datasets are often limited in size due to patient confidentiality and the issue of getting consent from patients. For this reason, the UNet was designed to work on limited datasets by focusing on capturing spatial information affectively.

Architecturally, a UNet is a "U"-shaped architecture consisting of 4 main components: encoder layers; decoder layers; a bottleneck and skip connections. The encoder layers make up the downsampling path and the decoder layers, the upsampling path. Originally, the UNet was exclusively made of convolutional layers, the Rectified Linear Unit (ReLU) activation function, max-pooling and transposed convolutional layers. This has been changed and experimented with for different applications since. However, the basic skeleton of the network remains the same. Figure 3 shows the basic template of a UNet.

Downsampling is performed through a number of encoder layers. Each layer takes an input and passes it through a convolutional layer with a $3 \times 3$ kernel. In this first convolution, the number of channels is increased, typically to a value which is a power of 2. The output of this convolution is passed through a ReLU activation function

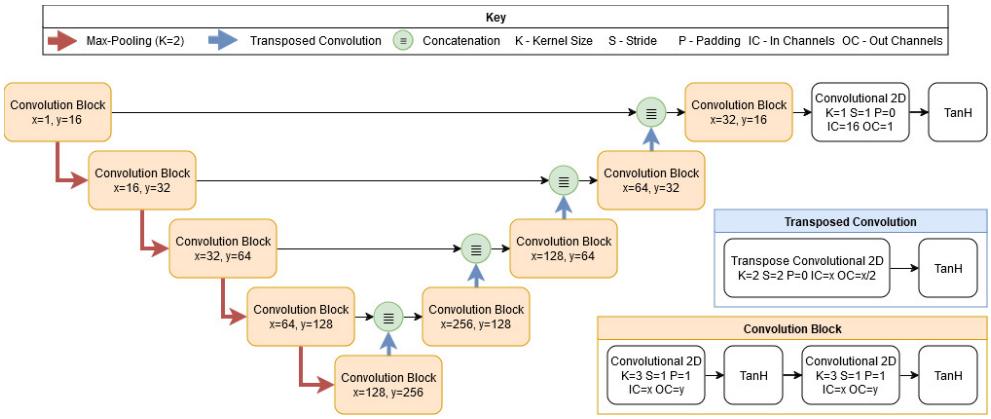



Figure 3: An example of a 2D UNet

before being passed through a second $3 \times 3$ convolution. The second convolution will maintain the current number of channels. Then a final ReLU activation function is applied. Following this final output of the encoder layer. This output then has a $2 \times 2$ max pooling operation applied which downsamples the output to half the resolution for the next encoder layer. As a result, each encoder layer has more channels but a lower resolution than the previous, allowing deeper encoders to assess features more holistically.

Following downsampling, the features from the deepest encoder are passed to the bottleneck. The bottleneck connects the downsampling path to the upsampling path via a number of additional convolutional layers to capture the more abstract features in the data. A bottleneck layer looks quite similar to an encoder layer without the need to apply a max-pool operation after computation. The bottleneck primes the data for passing to the decoder.

Upsampling takes place following bottleneck processing. This typically doubles the resolution of the data. Each decoder has a corresponding encoder. The input to the encoder, following upsampling, is concatenated with the output of the corresponding encoder before being processed by the decoder. This helps to maintain features across the network which may have been lost from downsampling. To align the resolution of the decoder input and encoder output, the encoder output was cropped to match the same dimensions as the decoder input. As such, the input channels for each decoder is double the output of the corresponding encoder, while the number of output channels remains the same.

Other approaches add padding to convolutions to maintain resolution throughout each encoder and decoder. Cropping is not required when this approach is taken. Each convolution can be padded to preserve the resolution of the data. The padding

applied can be defined as $\frac{\alpha-1}{2}$ where $\alpha$ is the kernel size.

Finally, a convolution is applied to the output of the highest-level decoder. This is typically a $1 \times 1$ kernel which applies final tweaks to the network output.

[22] likely chose to use a UNet for its ability to work on datasets of limited size, since research was conducted on the aforementioned "Motion Artifact Contaminated fNIRS and EEG Data" dataset. The work takes a standard UNet and adjusts the encoder and decoder layers. Instead, they are replaced with Multi-resolution Pooling (MRP) layers. MRP layers are designed to identify and emphasize key points of information. This is used on each encoder and decoder layer meaning that, at each step, the network works with multiple different resolutions to produce an output. This can be effective but it also requires a large number of parameters which can make the process slow and not fit for real-time application.

The innovation of this research was the combination of a number of factors. Firstly, the paper takes a standard FPN but adds a UNet inspired bottleneck. The purpose of the bottleneck is to give a more holistic view of the data. The bottleneck is to create a link between the encoder and decoder path at the lowest point. This is when the data is at the lowest resolution throughout the network, giving it a more holistic view of the data allowing it to capture high-level features. Analysis of high-level features can enhance generalisation and lead to more stable and consistent outputs.

Secondly, the paper takes a standard FPN but swaps out standard Convolutional Neural Network (CNN) layers for Self-Organised Operational Neural Network (Self-ONN) layers. Self-ONN layers are an extension of Operational Neural Network (ONN) layers which are an extension of CNN layers.

Traditional CNN layers apply fixed kernels to data which are learned through training. ONN were introduced to overcome some of the limitations of CNNs. Although CNNs are effective at tasks such as image recognition, they are less effective with diverse datasets and can fail to generalise well [25]. ONNs tackle some of the issues with CNNs by incorporating mathematical functions in place of fixed kernels [26]. These functions add a non-linearity to the way data is handled in the network.

However, ONNs still rely on predefined functional sets which limits their adaptability. Self-ONNs add to this by self-organising its use of non-linear functions [27]. This means that optimal functions do not have to be defined in advance but can instead be found during training. Research has found Self-ONNs to provide superior performance over ONNs and CNNs.

Finally, the research adds an attention to each skip connection in the network.

This attention should provide the decoder with information regarding the significance of each section of the signal, improving overall performance.

Another thing of note is the use of CycleGANs on the dataset from [17]. Due to the limited size of the dataset, it is questionable whether the dataset is sufficient. Before augmentation, there are only around 135 samples, of 4 seconds, per participant. This does not change dramatically following the suggested augmentation process. CycleGANs typically require a large amount of data in order to accurately discriminate between domains and reproduce the patterns. The limited nature of this dataset likely inhibits the CycleGAN from achieving a high level of accuracy.

In addition, a CycleGAN has been used on a matched dataset. The main strength and appeal of a CycleGAN is its ability to work well on unmatched data. Using a CycleGAN in this case complicates the process when working on a dataset that would have been trainable using standard training methods. It is possible that using a CycleGAN was seen as a method to further augment data through the use of 2 generators producing simulated outputs. This is because cycle-consistency loss requires the outputs of each generator to be passed through the other and used to calculate gradients. In theory, this should create additional dataset augmentations.

However, the reasoning behind using a CycleGAN is not explained from that point of view and seems an unnecessary over complication. Nevertheless, it does provide an opportunity to use multiple EEG datasets to potentially improve the result of training.

Other research into artifact removal in EEG has also turned to the use of machine learning. [28] used machine learning to remove Electrocardiogram (ECG) contamination from EEG signals. A Random Vector Functional Link Network (RVFLN) was suggested for this task due to the non-linear and non-stationary artifacts caused by ECG. The RVFLN is a relatively simple network based on standard Single Hidden-layer Feed-forward Network (SLFN). However, it overcomes issues with SLFNs, such as the sensitivity of the learning rate and slow convergence, by randomly assigning hidden node parameters based on certain probability distributions. The network also features some connections between the input and output layers. These connections aim to regularize enhancement features by providing reference to the original input in the output nodes.

This network was trained on a semi-simulated dataset [29]. The outcome of this research produced an output with an SNR of 68.1117 dB which is an improvement of 14.6117 dB over the best performing alternative, Complete Ensemble Empirical Mode Decomposition with Adaptive Noise (CEEMDAN+AF). Of the 6 methods compared

| Method | SNR (dB) |
|---|---|
| EMD + AF [30] | 51.4 |
| CEEMDAN + AF [30] | 53.5 |
| ANFIS + PSO [31] | 21.9 |
| [28] method | 68.1 |

Table 1: SNR values quoted in [28]

| Method | IQ |
|---|---|
| Infomax | 0.341 |
| ICA | 0.350 |
| RLS notch filter [32] | 0.331 |
| [28] method | 0.330 |

Table 2: IQ values quoted in [28]
(lower is better)

to in this study, only 3 of them have an SNR quoted (see table 1) and the other 3 are compared to using Information Quality (IQ) (see table 2). This makes comparison of the provided methods complicated and the results unclear. However, it provides the highest of the SNR scores and the lowest IQ score, meaning it is the most effective method based on both metrics, as a lower IQ score indicates less randomness. This demonstrates the usefulness of machine learning when applied to EEG signals for the purpose of artifact removal.

While encoder-decoder architectures are commonly used for handling EEG signals, Long Short Term Memory (LSTM) networks also make a regular appearance in the world of signal processing [33]–[35]. LSTMs are a form of Recursive Neural Network (RNN) which are designed to handle sequential data. This is particularly relevant when working with time-series data, such as EEG signals. LSTMs use memory cells to manage the flow of information over time. This allows it account for temporal dependencies. However, as the dataset commonly used for EEG motion artifact removal contains artificial motion contamination, the typical indicators which may be picked up by such a network are unlikely to be present in the dataset.

LSTMs inherently require greater memory usage than their encoder-decoder counterparts as well. This is due to the memory cells required to retain information between cycles. As a result, processing overheads can be massively increased and running an instance of the network for each electrode would become a challenging and expensive task. As EEG headsets range in the number of electrodes used, with numbers ranging into the hundreds, this makes LSTMs a less practical choice when looking at scalability and cost-effectiveness.

LSTMs have the benefit of previous context when analysing a signal. This is because of the memory cells which preserve relevant information. On the other hand,

data passed to UNets and other encoder-decoder architectures, lack the context of previous values. Outputs from a UNet, therefore, are less likely to align with previous outputs meaning that all data processed by a UNet for this task must be used in a windowed manner. This might not be ideal when trying to assess engagement over an extended or specific period of time.

In contrast to UNets, LSTMs have a linear architecture. Downsampling in the UNet allows lower frequency variations to be assessed more closely. Motion contamination is mainly of lower frequency which gives a UNet a theoretical advantage in this area. The use of downsampling and assessing the signal at a series of different resolutions, and in a less computationally expensive way, gives the UNet and other encoder-decoder architectures the edge over LSTMs for this particular use case.

The task of removing motion artifact contamination from EEG signals can also be viewed as a domain transformation task. A lot of research has been conducted into transforming images from one domain to another [36]–[38]. For example, converting images of cats into images of dogs. Such domain transformations focus on maintaining background information and only making the necessary changes to the target of the transformation. In the case of transforming images of cats into images of dogs, this means the colour of the fur will likely be maintained across the transformation. For EEG motion artifact removal, this is an essential characteristic as relevant features within the noise need to be maintained whilst exclusively removing the noise.

[39] recently found excellent results with the UVCGAN - a CycleGAN using a encoder-decoder architecture for the generator and a patch discriminator. The UVCGAN takes a standard UNet and adds a Vision Transformer (ViT) [40] to the bottleneck. In the bottleneck, the 2D image is flattened and positional encoding is applied before being fed through the transformer encoder. The output of the encoder is then reshaped back into the original 2D shape. In the research, it is hypothesised that the skip connections in the UNet pass high-frequency information whereas the transformer encoder acts as a method of learning pair-wise low-frequency relationships.

Also introduced by the paper are a gradient penalty for the discriminators to prevent the discriminators learning too quickly and a pre-training method to initialise the weights based on an inpainting task to achieve better performance. When compared to a regular CycleGAN, the UVCGAN outperformed the CycleGAN on Fréchet Inception Distance (FID) and Kernel Inception Distance (KID) scores significantly. The drawback of the UVCGAN is that it takes 1.5 times as long to train as a regular CycleGAN (60 hours instead of 40 hours) and is computationally more complex with almost 3 times

as many parameters. That being said, it is an important and relevant example of how domain transformation has improved and could be applied to one-dimensional tasks.

As with the UVCGAN, [41] found success by using a transformer-based architecture as the bottleneck of a UNet, further proving the efficacy of the concept. State-of-the-art results were achieved for medical image segmentation, with average improvements in accuracy on the BTCV multi-organ CT dataset of up to 13.71% (with an average dice score of 88.39%) over a standard UNet. This is credited to the transformer encoder facilitating the use of global contextual information which could not be achieved with standard convolutional layers which focus solely on a local area.

Transformers, introduced by [42] are a type of neural network architecture specialising in analysing sequential data. The core innovation of the architecture is self-attention, a mechanism used to identify the significance of different parts of the input. Self-attention allows Transformer-based architectures to analyse long range-dependencies and inherently consider the order of trends. This has the potential to offer significant benefits over existing methods for the processing of motion contaminated EEG signals as it should allow the network to identify motion contamination more effectively than those that only use convolutional layers. Convolutional layers are not able to account for order and long-range dependencies but instead focus on a set window. While this is useful in many use cases, it is physically impossible for motion contamination in an EEG signal to precede the indications of impending motion contamination. Therefore, a transformer should be able to learn that certain trends indicate motion contamination and only apply any correction to the proceeding data.

Due to the small size of the available dataset [17], it is clear that a larger amount of data would be beneficial. Creating a large matched dataset would be time consuming. However, many datasets are available for EEG, some containing motion artifacts. For this reason, the ability to use unpaired data would prove useful for removing the motion artifacts. As a result, techniques that work on unpaired data need to be considered. Some of these methods have already been discussed and used by aforementioned research, for example [6], but fail to take full advantage of the benefits of the technique.

Introduced in 2014, Generative Adversarial Networks (GANs) are a technique devised to train neurzal networks on unpaired data [43]. GANs are based on game theory and have found great success in generating realistic data [44]. This is particularly useful for tasks in which a paired dataset is not possible. GANs have also shown to be useful for generating data to enhance other training datasets [45]. GANs can overcome

issues with small datasets by synthesizing realistic data based on the datasets. As such, techniques such as data augmentation may not be required. However, the usefulness of GANs is limited to continuous data and falls short when handling discrete data such as the generation of words [46].

The objective of any GAN is to achieve the Nash Equilibrium. Nash Equilibrium is a game theory concept which defines a state in a game in which no player can improve their strategy without the other player first changing theirs. This creates a situation in which each player's strategy is optimal and there is no incentive to deviate from the current strategy for any player. In the case of GANs, this applies to the generator and discriminator playing a game in which the generator is trying to fool the discriminator and the discriminator is trying to outsmart the generator. This state is achieved when the discriminator is consistently uncertain if it is processing real of generated data. In other words, it consistently gives a 50% chance that the input is real. This concept allows GANs to work and become stable. Achieving this equilibrium can be challenging and requires the fine tuning of hyper-parameters.

An area in which GANs excel is image generation and image domain translation. With the use of a GAN, images of horses can be translated to pictures of zebras or pictures of men can be translated into images of women and visa versa. In these scenarios it is not possible to create a matched dataset. As a result, GANs provide opportunities to train networks to complete tasks which would otherwise not be possible.

In addition, using a training method which allows for unmatched datasets has the potential to be beneficial for motion artifact removal from EEG as it means that collecting a matching dataset is not necessarily required. Furthermore, while a dataset is available, it is limited by the use of multiple electrodes which are highly correlated but not an exact match. The size of the dataset is also limited. As a result, it is useful for validation but the potential to use multiple datasets for training could result in much more capable models.

A GAN works through the use of multiple networks. The first, referred to as the generator, is responsible for the translation task. Generators are the output of GANs and the part which is being trained to complete the desired task. A discriminator is the second network in a GAN. Discriminators are responsible for telling the difference between a generated output and an original one.

Generators in GANs are typically variations of convolutional networks. This can span from a network which do standard up-sampling to networks such as UNets and

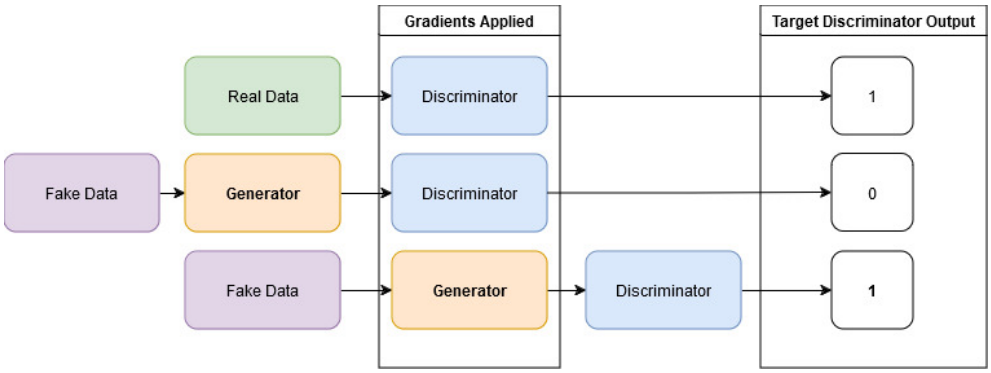



Figure 4: GAN Training Example

FPNs and other more complicated networks.

Discriminators vary according to the task. However, the input to the discriminator should be of the same resolution as the output of the generator. It is important that the discriminator is balanced with the generator to ensure the stability of the training. If the discriminator is too weak or too strong, the training of the generator will suffer. If the discriminator is too weak, the generator will fail to learn complex features and will suffer from general poor quality generation. If the discriminator is too strong, the network is likely to suffer from issues such as vanishing gradients and mode collapse.

Training a GAN requires 3 steps. As an example, imagine a scenario in which an image of a cat needs to be translated into an image of a dog. Both the discriminator and the generator have to be trained. In order to train the networks, labels have to be assigned to the genuine and generated data.

Commonly, 1 is used for genuine data and 0 is used for genuine data. However, this is a recommended starting point and should be changed depending on the activation functions used in the network. For example, if a Sigmoid or Softmax activation function is used, 0 and 1 should provide reasonable results. On the other hand, if an activation function which does not give an output between 1 and 0 is used, such as Tanh, adjusting these values may be required to get the best results. The way loss is calculated in a GAN is visualised in figure 4.

$$L_{discriminator} = \frac{1}{m}\sum_{i=1}^{m}\left[log\left(D\left(y_i\right)\right) + log\left(1 - D\left(G\left(x_i\right)\right)\right)\right] \tag{1}$$

To calculate the discriminator loss, an image of a dog would be passed through the discriminator. The output of the discriminator would then be compared to the genuine label, in this case 1. Secondly, the images of cats would be passed through the generator. The output of the generator will be passed through the discriminator and the value will be compared to the value for generated images, in this case 0. These

losses are both applied to the discriminator so it learns the difference between genuine and generated images. An example of discriminator loss can be found in equation 1.

The loss for the discriminator can change or be adjusted depending on the task. Independent of the loss function used, the premise remains the same. It has to be able to differentiate between genuine and generated inputs.

$$L_{adversarial} = \frac{1}{m} \sum_{i=1}^{m} log\left(1 - D\left(G\left(x_i\right)\right)\right) \tag{2}$$

Once the discriminator has been trained, the generator can use the discriminator to improve itself. In a GAN, the generator is trained to trick the discriminator into classifying its outputs as genuine rather than generated. For this example, an image of a cat would be passed into the generator. The output of the generator would then be passed through the discriminator. Discriminator output will then be compared to the value assigned for genuine dog images. The loss is then applied only to the generator meaning that the gradients calculated for the discriminator are solely used to update the generator. The original method for calculating generator loss can be found in equation 2 where 1 is the value assigned for genuine images.

$$L_{adversarial} = \frac{1}{m} \sum_{i=1}^{m} log\left(D\left(G\left(x_i\right)\right)\right) \tag{3}$$

Research has shown that minimising this function for the generator can lead to early vanishing gradients. As a result, maximising the output of the discriminator, using the loss function in equation 3, was suggested as it solves the issue of early vanishing gradients and provides stronger gradients earlier in training [43].

Whilst GANs are difficult to balance, this modified loss function generally leads to more stable training. Other methods such as Wasserstein GANs (WGANs), spectral normalisation of the discriminator and gradient penalties for the discriminator, have all been suggested as ways of increasing the stability of training Generative Adversarial Networks. WGANs implement a different type of loss for the GAN, replacing it with Wasserstein distance which makes the training process smoother by giving a different measure of distance between distributions.

GANs offer an elegant solution to the problem of working with unmatched data. Basic GANs suffer from many drawbacks, which need to be overcome to work reliably. Firstly, GANs suffer from training instability. Balancing the generator and discriminator can be a challenge because an imbalance can lead to many issues such as mode collapse, vanishing gradients and training divergence.

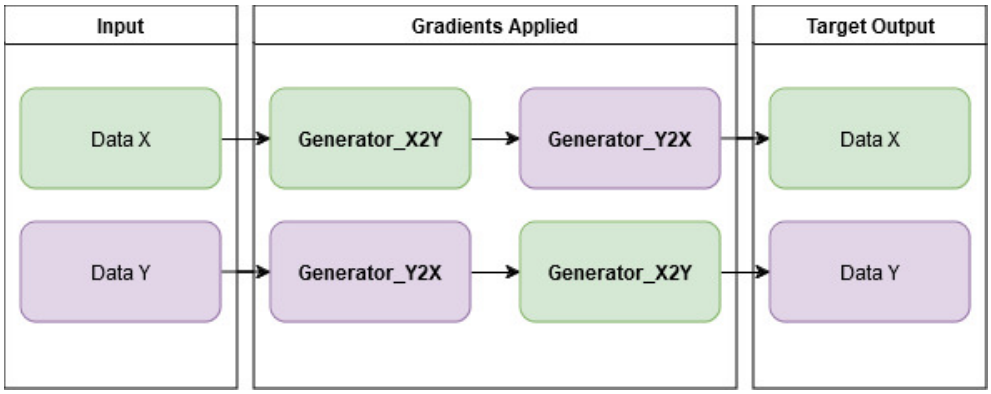



Figure 5: Cycle consistency loss example

Mode collapse occurs when the generator of a GAN learns to ignore inputs. It has been shown that GANs can be trained to a stage where the generator outputs will have low diversity regardless of the input. In this instance, the generator network learning to create one single output that effectively tricks the discriminator into believing the output is genuine. Mode collapse can be prevented by methods such as minibatch discrimination and, if the output should be representative of the input, CycleGANs.

GANs are also not trained to imitate their input. They can be trained to receive noise to produce an image of a particular item. This does not work in the application of removing motion artifacts from an EEG signal as it requires the features of the signal to be maintained and represented in the output. A solution to this problem is the CycleGAN which uses 2 GANs and additional loss functions to ensure that information is maintained in the output of the generator.

CycleGANs innovate on the original GAN in a number of key areas. Like regular GANs, CycleGANs allow for the training of a network on unmatched data. However, CycleGANs differ from standard GANs in complexity.

The fundamental concept behind a CycleGAN remains the same as with a standard GAN: generators are trained to complete a generation or translation task on unpaired data through the use of discriminators. At their core, CycleGANs are 2 standard GANs tied together by loss functions.

As for the previous example of translating images of cats into images of dogs, there would also be a secondary GAN with the aim of translating images of dogs into cats. In this instance, we end up with 2 generators and 2 discriminators both accomplishing different tasks: one generator responsible for translating images cats into dogs; another generator for translating images of dogs into cats; a discriminator for telling real dog images from fake dog images and a discriminator for telling real cat images from fake ones.

To begin with, both GANs are trained in much the same way as before. They are treated independently from each other as the discriminator and generator are trained. The innovation of the CycleGAN, is cycle consistency loss. The theory behind cycle consistency loss is, in order for features to be maintained in images, it should be possible to translate an image of a dog which has been translated from an image of a cat back into the original image. In other words, a genuine image should be able to be passed through the corresponding generator, then through the opposite generator, and the output should be the original image. This creates a link between the 2 GANs which forces the generators to produce results which are largely similar to the original input. This loss is applied to both generators for both configurations as shown in figure 5. Cycle consistency loss can be formulated as below.

$$L_{cycle}(G,F) = \frac{1}{m}\sum_{i=1}^{m}[|G_{y2x}(G_{x2y}(x_i)) - x_i|] + \frac{1}{m}\sum_{i=1}^{m}[|G_{x2y}(G_{y2x}(y_i)) - y_i| \quad (4)$$

Cycle consistency loss is a useful tool to preserve data throughout a domain transform. A multiplier is commonly used to control the influence of the cycle consistency loss over the training process. Commonly, 10 is the multiplier used multiply the cycle consistency loss in the original paper and is a recommended starting point for any other work.

As training progresses, cycle consistency normally improves to a point at which it becomes less influential than adversarial loss. However, it remains a useful tool throughout training to ensure consistency following domain transformation.

Identity loss is another loss introduced for use in CycleGANs. Whilst not a requirement of CycleGANs, it can help to reduce noise and maintain features through the generators. Identity loss aims to teach the generators to only translate the correct input to another domain. Take the example of images of dogs and cats again. If a picture of a cat is placed in the generator designed to translate pictures of dogs into cats, the expectation would be that no translation is applied to the image. In other words, the output of the generator should be the input image. This is exactly what identity loss aims to accomplish. By passing an image through the incorrect generator and calculating the difference between the input data and output data as the loss. An example of identity loss can be found below.

$$L_{identity} = \frac{1}{m}\sum_{i=1}^{m}[x_i - G_{x2y}(x_i)] + \frac{1}{m}\sum_{i=1}^{m}[y_i - G_{y2x}(y_i)] \quad (5)$$

Identity loss is then combined with adversarial loss and cycle consistency loss to create the complete loss function for the generators. The influence of identity loss is controlled with a multiplier. The most common multiplier of identity loss is 5. As a rule, the identity loss should be half the value of cycle consistency loss [47].

CycleGANs are not without their limitations and suffer from many of the same issues as regular GANs. Finding balance in a standard GAN can be a challenge. When handling 2 at the same time, it further complicates balancing the hyper-parameters. While there are methods of overcoming this, such as a grid search, it still takes much longer than traditional methods.

Due to their inherent complexity, CycleGANs are also computationally expensive to train. Having to train 4 networks rather than 1 naturally increases training time and computational requirements. This can lead to more expensive equipment being required and slows down iteration and testing.

On the other hand, the ability CycleGANs to work on unpaired data in an unsupervised capacity leads them to be a very useful tool despite the drawbacks. Especially, when tackling challenges where a matched dataset may be difficult to create.

A common issue with CycleGANs is discriminator over-fitting. This is when the discriminator quickly becomes too accurate and too specified to the training data. This overconfidence can lead to the generator not being able to keep up. As a result, the generator loss functions fail to adapt the generator in the right way and results can stagnate early or even get worse.

Designing a discriminator to match the strength of the generator is a potential solution to this problem. If discriminators are quickly over-fitting, simplifying the discriminator design is a possible solution to the problem. This can also lead to other problems, however, as a weak discriminator will lead to poor quality training of the generators. There are many tested discriminators for various tasks [48], [49] which act as useful starting points for new research. Using variations of these and adapting them to suit the needs of a task can be a good starting point. Then other methods can be used to adjust the strength of the generator.

The choice of optimiser can also control the strength of the discriminator. Adam is a commonly used optimiser for both generator and discriminator training (for example in [50]). However, this can lead to the discriminator becoming too strong too quickly. One method of overcoming this is to use different optimisers designed to prevent over-fitting [51]. An example of this is using a Stochastic Gradient Descent (SGD)

optimiser which typically optimises networks more slowly than the Adam optimiser, allowing the training of the generator to keep up. Other methods include adding a gradient penalty to the discriminator loss function which prevents the discriminator from learning too quickly [52].

Adjusting the label values for real and fake data is another method of accounting for discriminator over-fitting. Commonly, 1 is the label assigned to real data and 0 is assigned to fake data. This is useful for discriminators using a activation function on their final layer which normalises the output between the values of 0 and 1 (e.g. sigmoid or softmax). Nevertheless, selecting these values from the extreme ends of the activation function output values can quickly lead to the over-fitting of the discriminator. Moving these values slightly towards each other (e.g. for a sigmoid activation function: setting the fake label to 0.1 and the real label to 0.9) can prevent overconfidence in the discriminator. It is also important to ensure these values are correctly adapted to the activation function in use. For example, for a Tanh activation function, the values of -0.9 and 0.9 for fake and real data respectively would be more appropriate.

Pooling is another method used in CycleGANs [53]. The pooling of data is used to slow the learning of the discriminator so that it is not always being updated based on the most recent generator outputs. A pool typically consists of 50 outputs from a generator and each generator has its own pool. Each iteration, generator output is sent to the pool with a 50% chance of replacing an existing item in the pool. When training the discriminator, it pulls a sample from the corresponding pool rather than getting an output from the generator in its current state. This slows the learning of the discriminator to prevent over-fitting and improve generator training.

Changing the learning rate of the optimisers for the generators and discriminators is something which has been experimented with more recently [54]. This is another method which can help to balance an over-performing discriminator. Adjusting the learning rate of the generator to be slower can help it from adapting to the discriminator too quickly before the discriminator has been able to identify the differences between real and fake data. Likewise, reducing the learning rate of the discriminator can prevent it from over-fitting or becoming too confident too quickly.

Overall, it is evident that there is a large quantity of research on EEG signal processing. Recent advances in the removal of motion contamination from EEG signals have come from the use of machine learning techniques to accurately identify and remove motion contamination. However, there are gaps in existing research,

especially with preprocessing methods before network processing. In addition, new techniques which have found success in other areas have the potential to further increase performance.

## 4.4 Research Gaps

Following review of the existing literature, several key areas of focus can be identified. From the available research, there is limited attention on removing motion artifacts using machine learning while looking exclusively at the EEG signals. Existing research mainly focuses on either using the provided accelerometer data or complex training techniques. Using the provided accelerometer data by [17], as in [22], leads to the requirement of accelerometers on every electrode used. This leads to greater expense for healthcare providers and more discomfort for wearers from the added pressure on the electrodes. Ideally, the method should be able to work on a single electrode without matching accelerometer data to avoid this issue.

Focus from other research has also placed emphasis on complex training techniques for unmatched data, such as the CycleGAN [6] while working on a matched dataset. This can be viewed in 2 ways. Firstly, results would likely be improved by not using a CycleGAN to train on a matched dataset. Secondly, the use of a CycleGAN opens the door to using a larger amount of data and to use the original dataset exclusively for evaluation. Both of these options should be explored for an improvement in results.

Finally, the inclusion of a transformer encoder at the bottleneck of an encoder-decoder architecture has the potential to provide better results by looking at low-frequency relationships between data. The idea is to utilise the inherent ability of transformer encoders to learn relationships between segments of data with respect to order. The transformer encoder can be passed low-resolution data to identify and process larger trends in the data. These two benefits should improve on the performance of convolutional layers which do not respect the order of the data or long-term relationships between data segments. By using the approach of [39], [54], it may be possible to achieve greater accuracy and success through the use of a transformer encoder and its ability to recognise relationships between patterns.

# 5 Research Questions & Aims

## 5.1 Research Questions

Firstly, can the number of parameters in current solutions be reduced but still achieve similar performance? Some of the reviewed methods, [22] in particular, feature complex architectures with great computational expense. This will be assessed by analysing the temporal correlation between the network output and ground truth signals.

Secondly, would the inclusion of a transformer encoder in the network architecture enhance the removal of motion artifacts from EEG signals? As success has been found using variations of the transformer encoder in other areas [39], [54], this has the potential to be applied to EEG signals due to their ability to work with chronological data.

In addition, how effective are the existing preprocessing techniques used on the EEG data before being processed by the network and is there room for improvement? Existing research has shown preprocessing techniques which are impractical for real-time application or do not apply preprocessing on motion contaminated data. The use of preprocessing techniques that are practical for real-time use-cases has the potential to increase performance and lower computational cost.

Finally, would the inclusion of additional datasets and the use of unpaired training techniques improve results? [6] used a CycleGAN for training but used a matched dataset. This suggests that there is scope for improvement in results using additional datasets to train the network.

## 5.2 Research Aims

This research aims to address the outlined research gaps. The aim is to produce a network with fewer parameters than the existing methods. This will allow the network to run on inexpensive hardware, reducing the cost of implementing the results in places such as hospitals where many use cases could be found but budgets are limited.

In addition, the aim is to improve the network through the use of recent techniques. Such techniques include the use of a transformer encoder at the bottleneck of the encoder-decoder architecture to give the network an understanding of the relationship between different patterns in the data. This also comes with the benefit of the transformer encoder taking order into account which is important in time-series data.

Finally, the aim is to improve the dataset and dataset preprocessing such that the network requires fewer parameters. This will be accomplished through the use

of improved preprocessing techniques as well as the use of training techniques which allow for the inclusion of additional data.

# 6 Method

To achieve the aims and answer the questions outlined, it was decided that the first question to address should be that of the dataset preprocessing as it was essential for all additional questions. This should then be followed by the trialling of different techniques using standard back-propagation training on the matched dataset. This training method is significantly faster than training methods used for unpaired data and would allow for faster iteration on the network design. Finally, unpaired training would be tested to facilitate the inclusion of additional datasets.

In order to create and train neural networks efficiently, it was important to select a Machine Learning (ML) framework to proceed with. As TensorFlow and PyTorch are the most commonly used frameworks in the ML sphere [55], both were considered. Following some testing, PyTorch was selected over TensorFlow for its simplicity, documentation and community. TensorFlow appears to be used more in a production environment making it incredibly efficient but with a steeper learning curve. The benefit of using PyTorch was its low barrier to entry and its large adoption within the academic community.

In order to ensure repeatability, at the beginning of each test, PyTorch was given a seed of 15. This was to ensure that random values generated throughout training were consistent throughout trials and to ensure fair and repeatable results.

All training and testing was performed on a computer running Windows 11 with an AMD Ryzen 9 5950X CPU, 32 Gigabytes (GB) of Double Data Rate 4 (DDR4) Random Access Memory (RAM) and an NVIDIA GeForce RTX 3060 with 12 GB of Video Random Access Memory (VRAM). CUDA was used to increase the speed of training and testing.

## 6.1 Dataset Overview

As standard, the dataset procured by [17] was used to assess the accuracy of motion artifact removal techniques. The dataset for EEG contains readings from the motion contaminated and ground truth electrodes as well as X, Y and Z readings from the accelerometer attached to each electrode and a signal which indicated when the trial is in progress.

However, the electrodes and accelerometers were sampled at different frequencies. The accelerometers were sampled at 200 Hz while the electrodes were sampled at 2048 Hz. For the purposes of this research, the accelerometer data can be ignored. The aim was to produce a network capable of removing motion artifact contamination from EEG signals which can be applied in a cost-effective way. As accelerometers attached to each electrode is not a standard feature of EEG headsets, the use of the accelerometer data was not viable.

After reviewing the literature, it is clear that the best avenue forward in this area of research is through the use of machine learning. Classical algorithms do not provide suitable adaptability and require high levels of complexity. Machine learning provides an advantage from its ability to dynamically fit the problem, requiring less human input and higher accuracy. As a result a machine learning approach was taken to the problem.

Firstly, the "Motion Artifact Contaminated fNIRS and EEG Data" dataset was downloaded from PhysioNet and assessed. The claim of [18] is that the recordings of participants 12 and 15 are very poorly correlated and contain no actual brain data. To assess this, the dataset was loaded into a Jupyter notebook detrended using the SciPy library (scipy.signal.detrend [56]). The validity of all trials was then assessed by finding the Pearson Correlation Coefficient (PCC) between the ground truth and motion contaminated signals.

PCC is used as a performance metric for comparing one-dimensional temporal data such as signals. Using PCC, the expected output of a network can be compared with the actual output, where a value of 1 is an exact match and a value of -1 is the exact opposite. This offers a way of assessing the accuracy of the network output by direct comparison with the ground truth signal. The equation to find the PCC can be found in equation 6.

$$r = \frac{\sum(x - m_x)(y - m_y)}{\sqrt{\sum(x - m_x)^2 \sum(y - m_y)^2}} \tag{6}$$

While the PCC is a useful metric for this application, it is not a perfect solution. For example, it reports on the correlational error and not the absolute error. That means that, if the values being compared to each other are well correlated but offset by a constant amount, this will not be reflected by the PCC. This is not ideal as it means a signal which is offset from the target signal could still be considered highly correlated. However, this issue can be somewhat negated in this case through the choice of loss function in the network as loss functions typically focus on absolute error. This means

that, throughout training, the output should converge such that any offset created is negligible. As a result, the PCC offers a reliable measure of accuracy in this use case.

The PCC was calculated using the SciPy library (scipy.stats.pearsonr [57]). Following an analysis of the results, it was found that both trials 12 and 15 had an above average temporal correlation. For this reason, the claims that they should be excluded were ignored and they were kept in the dataset.

In order to accurately assess performance of any network produced against existing research, the data augmentation methods of [22] and [6] were both reproduced for use. These works are the biggest recent contributors to the machine learning approach to the issue of motion artefactual EEG signals.

However, there are some notable inconsistencies which need to be addressed. In both of these approaches, the dataset is augmented in different ways and then validated with the leave-one-out approach. Although this offers some metric by which to assess the performance of the network, it also creates an issue in which the results are not directly comparable. Whilst training on different augmentations of the dataset has the potential to provide better results, validation needs to be completed on similarly processed data in order for it to be a meaningful comparison.

$$OverlappingPercentage = \begin{cases} 0, & \text{if } \eta_{temp} > 0.5 \\ 10 \times (\lfloor (1 - \eta_{temp}) \times 10 \rfloor), & \text{if } \eta_{temp} < 0.5 \end{cases} \tag{7}$$

The way in which the augmentation differs is in overlap. Both methods handle windows of 4 seconds of data at a time. The difference is that both methods overlap these windows at different intervals to increase the amount of data and make up for the small sample size of the dataset. In [22] each window had a 50% overlap with the previous, which means that twice as many samples were created. On the other hand, [6] used an equation to determine overlap based on the correlation between the ground truth and motion contaminated signals in the given window (see equation 7).

This creates a problem when comparing the results of the networks. The earlier research, [22], claims a higher correlation between ground truth and network output than that of [6]. When looking at the way the dataset has been processed, however, it is clear that this should be the case. The leave-one-out approach to validation sees the network trained once for each trial, known as a fold. In this case, trial 1 might be left out while the network is trained on trials 2 - 23. The trained network would then be validated against trial 1. Here, the validation metric is the PCC. This is then repeated

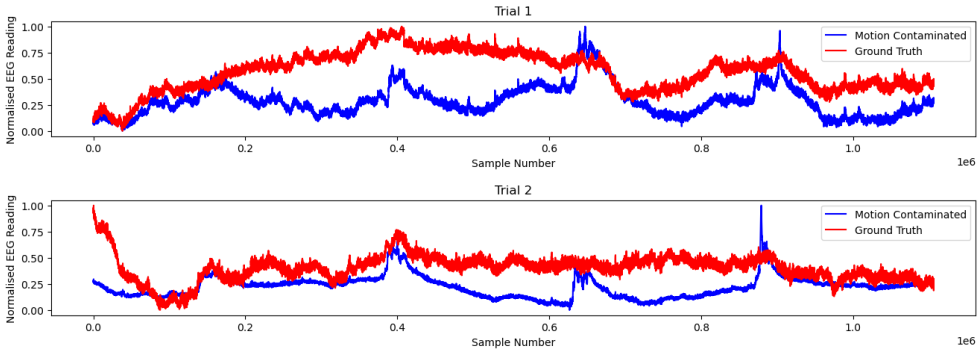



Figure 6: Normalised ground truth and motion contaminated signals from trials 1 and 2

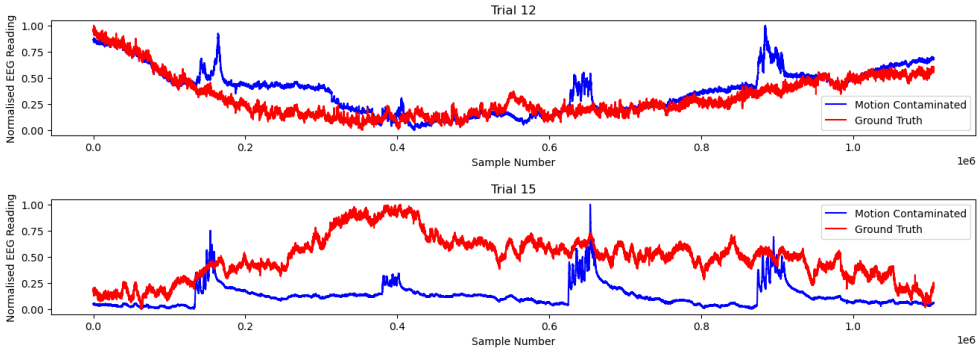



Figure 7: Normalised ground truth and motion contaminated signals from trials 12 and 15

for trials 2 - 23 and the average validation score is taken as the score for the network.

This is particularly useful for small datasets where all data might be required for training. In this instance, though, it is of note that the motion contaminated signal is not pure motion contamination.

For example, figures 6 and 7 clearly show that there are 4 different regions of motion contamination in the motion contaminated signal but they are separated by sections containing no motion artifacts. As [6] dynamically adjusts the window overlap based on the PCC, this means that the dataset used in [22] contains a different ratio of significant motion contamination for training and validation. The network could be expected to perform much better on less contaminated data as the PCC between the ground truth and motion contaminated signal at those points should be much closer to 1.

From this, it can be seen that the network would perform well simply by not adjusting the input. As a result, the claims of accuracy from either paper (a PCC of 0.905 for [22] and 0.872 for [6]) are not directly comparable.

To compensate for this, each dataset used had a second set used for validation. This set has a fixed overlap, of 50%, and was only used for evaluation. This provided

a means of assessing the dataset augmentation method to achieve the best results.

## 6.2 Dataset Augmentation

### 6.2.1 [22] Dataset Reproduction

The first data to be reproduced was that of [22]. As previously mentioned, this research also used the accelerometer data from the motion contaminated electrode. The requirement for accelerometers attached to each electrode is not one that is supported by the research aim of keeping the system as low cost as possible. Therefore, when reproducing the data augmentation from [22], only the electrode values were used. In the original work, the EEG readings were downsampled from 2048 Hz to 256 Hz. This downsampling was done to reduce the sampling rate of the electrodes to a similar rate as the accelerometers (200 Hz). As the accelerometer and EEG data were used concurrently, they needed to have a similar number of data points. While this was not the case for this work, only top-of-the-range EEG headsets are capable of sampling at 2048 Hz. Many cheaper headsets sample at much closer to 256 Hz. As a result, this downsampling is still beneficial to the aims of keeping implementation costs low.

To downsample the data, the BrainFlow [58] library was used using brainflow.data_filter.DataFilter.perform_downsampling. This was set to use the median value of the results.

Then each of the 23 trials was trimmed because the dataset contains extra readings at the start and end of each exercise. This relevant data is marked by the presence of a trigger signal which was used to extract the desired EEG data. Data was read from the provided Comma Separated Value (CSV) files using the built-in Python CSV library. Sections of the signal considered not a part of the trials were noted by a 0 in the fourth column of the CSV files. Using the Python CSV library, files are read row by row. Therefore, as the file is read in, any row containing a 0 in the forth column was excluded. The sections of all trials containing a 1 in the fourth column are continuous. As a result, different sections of the trial were not unknowingly combined. This method was used for the creation of each dataset augmentation.

Following this, the data was detrended using a polynomial fit. This is done to remove baseline drift which can occur during signal recording for several reasons largely related to the condition of the electrodes. For example, a change in electrode temperature or the electrodes moving on a subject's head can all cause baseline drift.

The polynomial fit was found using Python's Numpy library. The fit was first found

using numpy.polynomial.polynomial.Polynomial.fit [59]. Then, this fit was subtracted from the signal to detrend the data.

Baseline drifts are largely unpredictable and do not align across electrodes meaning that the ground truth and motion contaminated signals do not share the same baseline drift. If the neural network had to also take these wanderings into account, it would create confusion from random additional patterns. Hence, the researchers in [22] opted to remove this by fitting a polynomial of order 20 along each trial and removing it from the data. This was applied to both the ground truth and motion contaminated signals.

Following this detrending, the signals were split into windows of 4 seconds (1024 samples following downsampling) with a 50% overlap. Once separated, the new segments also underwent a polynomial detrending process. A polynomial of order 20 was fitted to all ground truth signals and removed. For motion contaminated signals, the process was less straightforward. Segments containing motion contamination were detrended with a polynomial of order 3. On the other hand, segments that did not contain motion contamination were once again detrended with a polynomial of order 20.

However, this decision cannot be applied in practice as there would be no way of differentiating between these signals. Arguably, the same treatment should be applied to the motion contaminated signal regardless. However, for the sake of matching the previous method, this was applied. The definition of motion contaminated segments was left ambiguous by the paper. As a result, sections with a PCC below 0.5 were considered motion contaminated. The decision to use 0.5 as the cut-off was not arbitrary. It was used as this was the value selected by [6] to increase window overlapping. For a consistent comparison, the same value was used here.

Finally, environmental noise, caused by power lines, was then removed from the signals using a IIR-based notch filter. EEG is very sensitive to background noise from sources such as power lines and it is necessary to remove them for proper data preparation. In this case, 50 Hz noise was removed. This was achieved using Python's SciPy library (scipy.signal.irrnotch [60]).

Then each segment was normalized using z-score normalisation followed by range normalization between 0 and 1. This is in accordance with the original method. However, the z-score normalisation does appear to be redundant as it is a linear transformation. This completed the recreation of the dataset augmentations used in [22].

### 6.2.2 [6] Dataset Reproduction

To reproduce the dataset augmentations from [6], a series of similar steps were taken. First, the data was downsampled to 256 Hz from 2048 Hz once again using the BrainFlow library and with the operation set to median. In the original research, the decision was also taken to undertake this downsampling step. However, in this case the accelerometer data was not used. It is likely that this was done to reduce model parameters as a 4 times smaller input would dramatically decrease computational complexity. Nevertheless, this is purely speculative as the reason for this downsampling is not stated.

The downsampled signals were then reduced to 4 second samples (1024 data points). In this case, only the ground truth signal was detrended. This was by fitting and removing a polynomial of order 10 to the ground truth segments. To implement this, the SciPy library's polynomial fit function [59] was once again used to find the fit, then subtracted from the signal.

The same IIR filter as before was then used to remove power line interference at 50 Hz. This was implemented using the SciPy library's iirnotch function [60]. The overlapping percentage with the next sample was then determined by the aforementioned overlapping percentage equation.

### 6.2.3 Custom Dataset

Following the reproduction of both datasets, it was observed that less than 30% of the data in each data set contained high levels of motion contamination. As previously mentioned, any section with a PCC below 0.5 was considered to contain high levels of motion contamination. To account for this, a custom variation of the dataset was created.

To design the preprocessing for custom data augmentation, the previous methods were first assessed. The method used by [22] includes fitting a polynomial to the full set of data prior to windowing. This makes sense in an attempt to achieve the best results but is impractical in real-time applications where a large amount of data may not be available. As the intended application of this research is for real-time use, this step had to be dropped from the workflow.

In addition, [6] states not detrending the motion contaminated data in order to justify the use of the neural network. This claim seems largely redundant as the aim of the neural network is ultimately to make the output as accurate as possible. Therefore, detrending the motion contaminated data should give optimal results.


Figure 8: Window overlap based on PCC for custom data augmentation

To create the custom dataset augmentations, each trial was first downsampled to 256 Hz. This was done using the BrainFlow library's perform_downsampling method, with the operation set to use the median. This step was taken to match the other methods [6], [22] but also to ensure that the network can be used on lower-end EEG headsets which have a lower sampling rate.

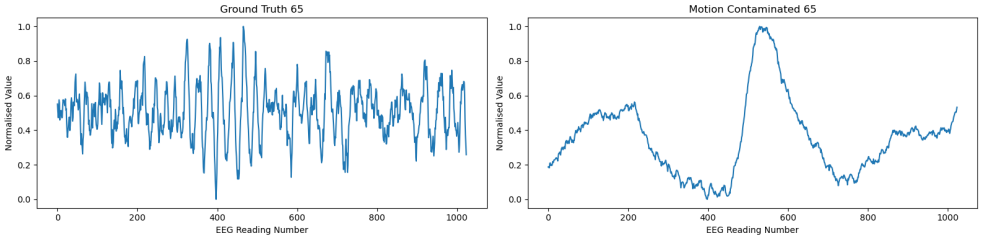
$$OverlappingPercentage = \frac{tan^{-1}\left(2\left(1 - n_{temp}\right) - 2\right) + 1}{2.2} \times 100 \tag{8}$$

Following this, the signals were segmented into 4 second windows. This is in alignment with the other studies. From this windowing process, the aim was to improve the ratio of motion contamination over the other 2 data augmentation methods. For this a new equation was created for the windowing process. The equation used can be found in equation 8, where $n_{temp}$ is the PCC below the motion contaminated and ground truth signals, which was used to place greater emphasis on sections with a lower PCC and less focus on sections with a higher PCC.

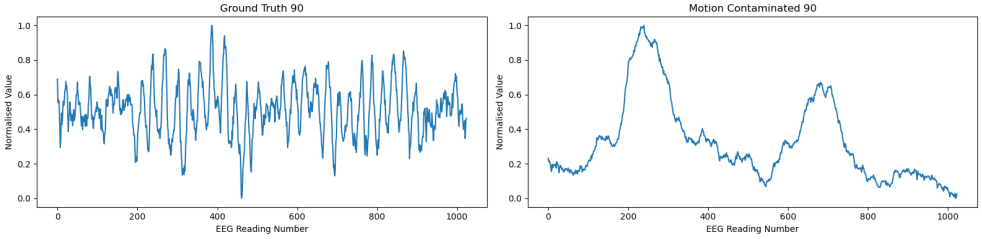
Visualisation of the resulting distribution can be seen in figure 8. From this technique, it can be seen that lower PCCs are gradually prioritised over higher PCCs. However, unlike the other distributions, it also accounts for results with reasonable PCCs and also gives some overlap on features with higher correlations. In theory, this should enable the network to learn motion contamination removal more clearly by also focusing on smaller deviations in correlation.

Following this step, a polynomial was fit to both the ground truth and motion contaminated windows and removed. This was implemented using Numpy library's polynomial fit function [59] to fit a polynomial along the signal and then subtracting the polynomial from the signal.

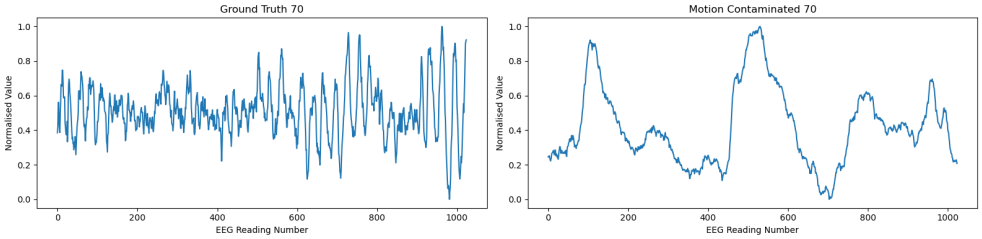
For this step, a polynomial with an order of 10 was used for the ground truth signals and an order of 4 was used for the motion contaminated signals. As the ground truth signal contains only important data, it is important to make sure the signal




(a) [22] Augmentation Example




(b) [6] Augmentation Example




(c) Custom Augmentation Example

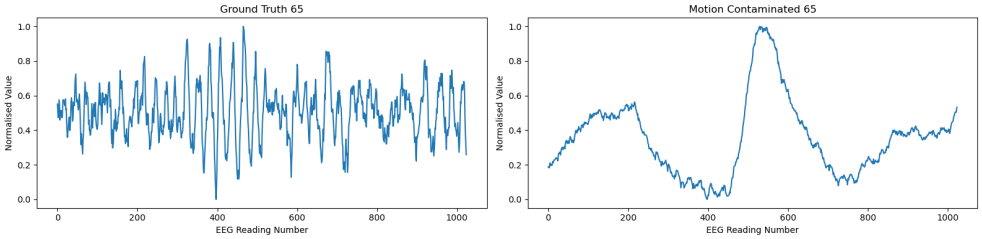
Figure 9: Examples motion contamination from trial 23 for each data augmentation technique

is as clean as possible. In addition, adjusting the motion contaminated signal too much has the potential to remove important data from the signal and emphasise the contaminated data. For this reason, a value of 10 was selected.

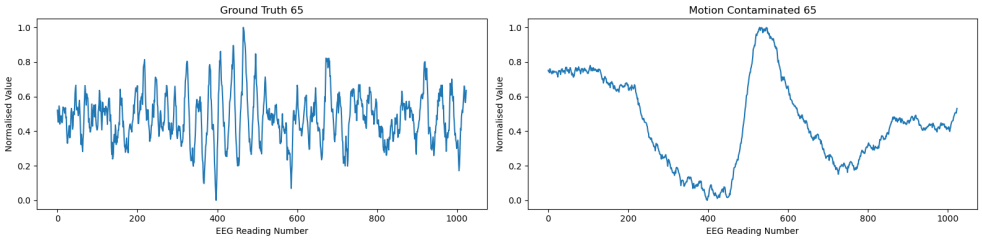
Following this, the same IIR notch filter was used on both channels. This is important for the removal of power line noise which is present in all of the trials in the dataset. The IIR filter was set to remove 50Hz noise from the signal as before and was implemented using the SciPy library's "iirnotch" function [60].

Finally, each window was range normalised between the values of 0 and 1. This is for the sake of processing the data to keep it in a form which can be passed through the network.

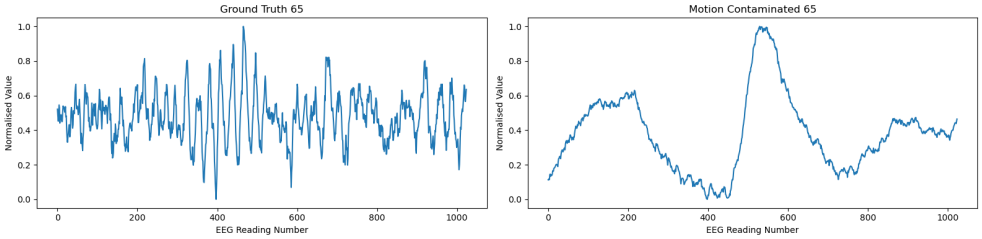
Figure 9 shows a motion contaminated window from each augmentation method. All of the examples are from trial 23 and show limited emphasis on the brain signal and much more on the motion contamination. It shows that, whilst there is not much difference between the preprocessing methods, the custom augmentations maintain larger details from the ground truth signal. This is more evident in 10 as the results are taken from the validation sets, meaning they are all from the same section of the




(a) [22] Augmentation Validation Example




(b) [6] Augmentation Validation Example




(c) Custom Augmentation Validation Example

Figure 10: Examples motion contamination from trial 23 for each data augmentation technique taken from the validation set for direct comparison

signal.

Figure 10 clearly shows the greater adjustments made by the custom pre-processing steps, while also displaying a clear need for additional processing to properly clean the signal. This displays and justifies the requirement of a neural network for additional processing, contrary to claims by [6] which suggested detrending the motion contaminated signal would remove the necessity for ML.

## 6.3 Dataset Selection

In order to proceed, it was necessary to use only one dataset. To determine the most useful data augmentation technique, they were all tested via training on a standard 1D UNet. A UNet is an encoder-decoder architecture with skip connections between like-sized encoders and decoders. UNets were originally used for medical image segmentation [24] and are the inspiration for many techniques in signal processing [61], [62] so it was chosen as a reasonable starting point to get informative results.

The designed UNet contains 5 layers and 677,105 parameters. Each encoder and decoder is made up of a convolutional 1D layer with a kernel size of 3 and a padding of 1 to maintain resolution. This is followed by a Tanh activation function. The convolution layer and activation function are then repeated once more to form an encoder or decoder.

For encoders, this is followed by a max-pooling layer to reduce the resolution of the layer by half for the following encoder or bottleneck. Decoder layers are preceded by transpose layers with a kernel size of 2 and a stride of 2 to double the resolution. Before decoder layers are processed, the feature maps (output) from the corresponding encoder are appended following the transposition of data. At the bottleneck, the same combination of 1D convolutions and activation functions, as used in the encoders and decoders, is used.

Finally, an additional convolutional layer with a kernel size of 1 is applied. This layer applies any final adjustments to the output and is standard in UNets. An additional activation function is then applied to the final output of the network.

For the first encoder, the number of channels is increased from 1 input channel to 16 input channels. As the resolution is halved down the encoder path, the number of channels is doubled. The opposite is true for the decoder path as the number of channels is halved as the resolution is doubled. The result is an output with the same number of channels and the same resolution as the input. Figure 11 shows a detailed diagram of the network.

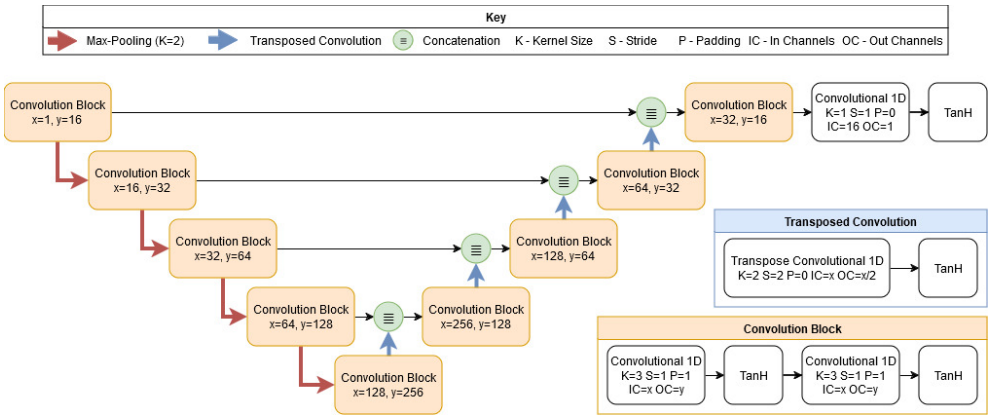



Figure 11: 1D Convolutional Network

This network differs from a typical UNet in 2 key areas. Firstly, a standard UNet is used for 2D tasks such as image processing. As processing EEG signals only requires 1D data, the network was adjusted accordingly. Secondly, the ReLU activation function [63] is typically used in UNets. ReLU is typically used for its computational efficiency and its ability to reduce the chances of vanishing gradients. However, both [22] and [6] use the Tanh activation function. This is likely for the purposes of keeping the data normalised between 0 and 1. It is also related to some requirements of the networks which will be discussed later. As a result, the Tanh activation function was used in place of ReLU for this task.

Trials were run on all variations of the data set for the sake of comparison. Weights in the network were initialised using Xavier normal distribution [64] with a gain of 1. The batch size used for training was 16, as per [6], and each fold was trained for 10 epochs to obtain a representative result.

The Adam optimiser was used with a learning rate of $2 \times 10^{-4}$ and beta values of 0.9 and 0.999. Adam is a commonly used optimiser for a wide range of tasks. Though Adam has been shown to have poor generalisation in some scenarios [65], it offers a way to get reliable results quickly, so it was chosen as a reasonable starting point.

Lastly, Mean Squared Error (MSE) was used for the loss function. Mean Absolute Error (MAE) (sometimes referred to as L1 loss) was also an option. However, MSE adapts more quickly to more significant errors. This means that the biggest errors in the signal output will be punished exponentially more. As the network removes motion contamination, large errors are a big issue. In addition, small errors are of less significance when the signal is evaluated. As a result, ensuring larger errors are minimised was crucial. MAE on the other hand, does not scale the loss exponentially for greater errors, theoretically making it a worse choice for this task.

Following the trials and analysis of the results, it was concluded that the proposed preprocessing and window overlapping methods should be used to proceed. This decision was made because the proposed method is capable of being applied in real-time which is essential for use with patients when quick feedback may be crucial. As a result, the preprocessing method described by [22] would not be feasible. Secondly, the results of the proposed preprocessing method significantly outperformed the method proposed by [6]. This meant that the decision to use the proposed method was straightforward.

## 6.4 Base Architecture Selection

Following the selection of a signal preprocessing method, it was clear that improvements needed to be made to the architecture of the network to produce optimal results.

UNets are a commonly used type of auto-encoder made up of an encoder-decoder architecture. FPNs are another example of an auto-encoder architecture [66]. First created for object detection tasks, FPNs have since been adapted to a range of different applications largely pertaining to object detection [67]–[69]. As it has been shown to be an effective method in detection tasks, its applications have the potential to expand to the detection and removal of motion contamination. For that reason, it was used as a comparison to the UNet architecture previously described.

To construct an FPN, another network is first required. This network is known as the backbone and can be thought of in a similar way to the encoder path in a UNet. The job of the backbone is to extract features at various different resolutions. For this reason, it is common for pre-trained models to be used as the backbone of FPNs, such as a ResNet [70]. In this sense, FPNs can be considered somewhat parasitical. The backbone, also referred to as the bottom-up pathway, creates the feature maps which will later be analysed by the FPN.

Following the feed-forward computation of the backbone, the FPN is used for feature extraction. The FPN makes up what is referred to as the top-down path of the network and functions in a similar way to the decoder path of a UNet. The top-down path should be constructed in such a way that each layer matches a layer in the backbone network. As such, skip connections can be created between like-sized layers in the bottom-up and top-down paths.

Unlike a UNet, FPNs do not feature a bottleneck. Rather the first layer of the top-down path is taken from the output of the backbone with a kernel of size 1 applied. Following this each layer is transposed to match the resolution of the next layer in the

FPN. From this point on, each layer in the FPN takes an input which is constructed of 2 parts. The first is the upsampled output of the previous layer in the top-down path. The second, is the output of the like-sized layer in the bottom-up path. These 2 values are added together to form the input of the network. This process takes place all the way until the top layer in the top-down path which outputs the same resolution as the input to the backbone.

Each layer in the top-down path applies a kernel to the input data, normally of size $3 \times 3$ for 2D data, followed by an activation function. This applies an additional processing step to the feature-extracted data, which enables deeper understanding from the network.

In order to construct an FPN comparable to the UNet tested previously, the encoder path was taken from the UNet and used as the backbone. This consists of 4 encoders, each half the resolution of the last but with double the channels. This was done to ensure that the backbone was fair and comparable for each network.

To form the top-down path, 4 decoder layers were created of matching size to each of the layers in the backbone. These decoders use the same kernels as those used in the decoders for the UNet for processing. This means that in each decoder layer, there are 2 kernels with a size of 3 and a padding of 1 applied. Each convolution is followed by a Tanh activation function.

This FPN differs from the UNet in multiple areas. The first is the lack of any processing at the bottleneck, which means that key details could potentially be missed. Secondly, the skip connections have a $1 \times 1$ convolution applied. This should improve feature recognition over the UNet due to the additional adjustment and processing step.

Due to the lack of a bottleneck and because each of the FPN decoder layers has half as many channels, the resulting FPN would have significantly fewer parameters than the UNet. In order to make the test fair, the number of channels was increased at each stage of the encoder and decoder. The first layer had 24 channels which was doubled for each layer further down in the network. The result of this is a network with 612,625 parameters which is much more closely comparable to the UNet's 677,105.

Parameter-wise this makes the FPN 9.5% smaller than the UNet. However, this is for 2 reasons. The first is the lack of any convolutions being applied in the bottleneck. This is the section of the UNet with the greatest number of channels which means a much higher number of parameters. Secondly, the layers in the top-down path of the FPN have half the number of input channels each. This is due to skip connections being

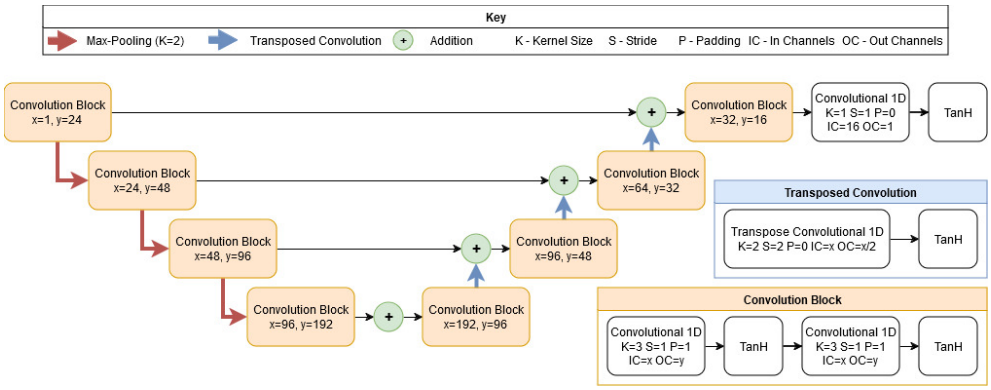



Figure 12: Convolutional FPN Used for comparison to the UNet

multiplied with the output of the previous encoder rather than being concatenated as in the UNet. Testing revealed that the FPN underperformed the UNet on all but one trial. Several potential solutions to this were considered. For example, by increasing the number of parameters in the network through the replacement of addition with concatenation in the top-down path. This would double the number of channels in the decoder kernels and give the network more information to work with in the top-down path. However, this would make the network larger than the UNet which is less than ideal in the venture for computational efficiency.

Highlighted by this result is the importance of the bottleneck for the identification of key features for processing by the decoder path. Therefore, it can be concluded that special attention should be given to the bottleneck and that the concatenation of encoder outputs to decoder inputs is preferable to summation. As a result, the UNet was used as the base architecture going forward.

## 6.5 Transformer Encoder Bottleneck

As the bottleneck was identified as a key component of the network for the task of removing motion contamination from an EEG signal, special attention was given to this area of the network. [39] suggested the use of a ViT for use in image translation tasks with the aim of identifying low-frequency features. The research shows significant performance gains from the use of a ViT at the bottleneck of the network with the additional use of positional encoding.

As this is not a 2D task, a ViT is not required. To utilise the potential displayed by the ViT in the work of [39], the bottleneck of the UNet was replaced with a standard transformer encoder as described in [42]. While [6] used attention mechanisms to improve the performance of the network, transformer encoders offer a number of benefits.

Transformer encoders have the ability to understand the order of the signal. For this use-case, the signal is split into tokens (small sections of chronological data) which is used by the multi-head self-attention mechanism to understand the relevance between the different tokens.

Tokens are created from an embedding matrix which is used to transform sections of the data into tokens. For each token, a query, key and value vector are created. For the query, the token is multiplied by a learned query matrix. The same is true for the key which is also the product of multiplying the token by a learned key matrix to produce a key vector for the token.

Masking is then applied to this matrix to ensure that each element in the sequence is only affected by those preceding it. This is very important in the case of signal processing as a signal cannot be retroactively impacted. To stop this, a mask is applied to set any points in this matrix which correspond to a key which appears later in the sequence than the query to negative infinity. Each of these values is then divided by the square-root of the number of dimensions in one key vector.

A Softmax activation function is applied to normalise the output matrix and give a probability distribution across the new matrix. This matrix is a measure of how each key attends to each query. This query is then adjusted further by a value vector which is computed for each embedding and adds context to the output.

Following this, the output is passed through a feed-forward network to refine the token representations. These encoders are normally used in a series, known as layers, repeating multiple times to further refine the output. The output is an encoded representation of the relationships between the different sections of the data. This is particularly useful when processing signals. For simplicity, the transformer encoder was implemented using the built-in transformer encoder offered by PyTorch [71]. The transformer encoder was set to have 8 attention heads and 4 layers.

Prior to the transformer encoder, learned positional encoding was added directly to the signal. Positional encoding was implemented similarly in the work of [39]. Learned positional encodings were concatenated with the existing data at the bottleneck and passed through a feed-forward layer with a Gaussian Error Linear Unit (GELU) activation function to embed the positional encodings and introduce non-linearity.

Another reason for the inclusion of GELU was also the use of Tanh activation functions within the network since Tanh normalises the values in the network between -1 and 1. An alternative to GELU would have been the ReLU activation function. The issue with this function is that it sets any value below 0 to 0. This had the potential

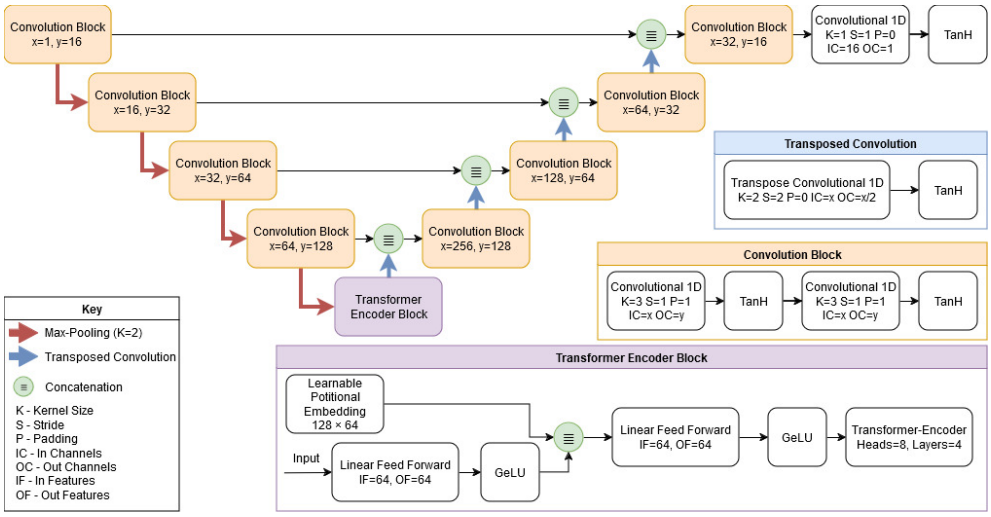



Figure 13: 1D convolutional UNet with a transformer encoder bottleneck

to remove half of the input data from the network. Therefore, GELU was a natural choice as it does not remove values below 0 from the data.

While it is common to use fixed values based on trigonometric functions for positional encoding [42], learned parameters were opted for as they could better adapt to the dataset and give better results overall. Due to the small resolution of the data at the transformer ($1 \times 64$), it was concluded that this small increase in parameters would lead to better overall results.

Following testing, it was found that the transformer encoder boosted the temporal correlation between the ground truth and network output, as expected. As a result, the transformer encoder was kept in the architecture. The architecture can be seen in figure 13.

## 6.6 Self-ONN Layers

One of the innovations of [6] was the use of Self-ONN layers. Self-ONN layers build upon the more traditional convolutional layers by swapping linear kernels for more dynamic ones. Historically, a convolutional kernel would be made up of linear values. For example, a kernel of size $3 \times 1$, as might be the case in a 1D network, would be made up of 3 values, one for each cell in the kernel. While this produces successful results, a common requirement of these layers is to use multiple in sequence to introduce non-linearity to the computations.

Convolutional layers are inherently homogeneous as a result which yields limited results. ONNs [72] and subsequently Self-ONNs [73] were introduced as a way of introducing nonlinearity to the flow of data through a network. This is achieved

by replacing the linear values of the convolutional kernels with a function. Initially, with ONNs, these functions would have to be defined manually. However, the later innovation of the Self-ONN allowed for functions to be dynamically created through back-propagation. These functions are made up of mathematical operators such as the trigonometric functions which enable create heterogeneous kernels.

Each Self-ONN layer has an associated Q-value. This defines the number of functions used within each kernel element, hence, controlling the non-linearity. Increasing the Q-value naturally increases the computational cost of the network, which is why lower Q-values are preferred. A Self-ONN layer with a Q-value of 1 is essentially a convolutional layer with no special variation applied. Therefore, in [6] a Q-value of 3 was used to introduce non-linearity without increasing the network parameters and computational cost too dramatically.

One of the added benefits of the Self-ONN layers is, due to their inherent heterogeneous nature, the requirement for using multiple in series to produce non-linear transformations and model the data more accurately is substantially reduced. Reducing network complexity as a result.

In an attempt to further improve the network, the structure of each encoder and decoder was changed as well as that of the final layer. For the encoder and decoder layers the double convolution was replaced with a single Self-ONN layer with the same kernel size and padding. The final convolution layer in the network was also directly replaced with a Self-ONN layer. All Self-ONN layers were assigned a Q-value of 3 as in [6].

A quirk of Self-ONN layers is the requirement to receive a value of between -1 and 1. This is due to some of the functions used within the network only working in this range. However, due to the previous use of the Tanh activation function, this was not an issue. The activation function was not changed following the swap from convolutional layers to Self-ONN layers.

Self-ONN layers were added to the model using the FastONN library [74] available for PyTorch. A network diagram can be seen in figure 14.

Following testing, it was found that the use of Self-ONN layers resulted in overall worse performance than that of the convolutional network variant. However, in [6], in order to protect against vanishing and exploding gradients, batch normalisation was added between each Self-ONN layer and activation function in the encoders and decoders.

Batch normalisation was added between the Self-ONN layer and the activation

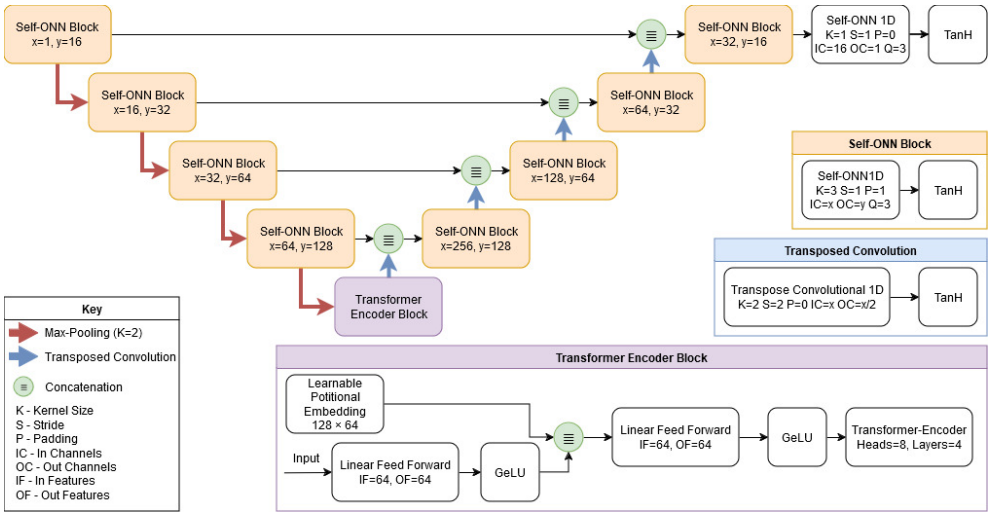



Figure 14: 1D Self-ONN UNet with a transformer encoder bottleneck

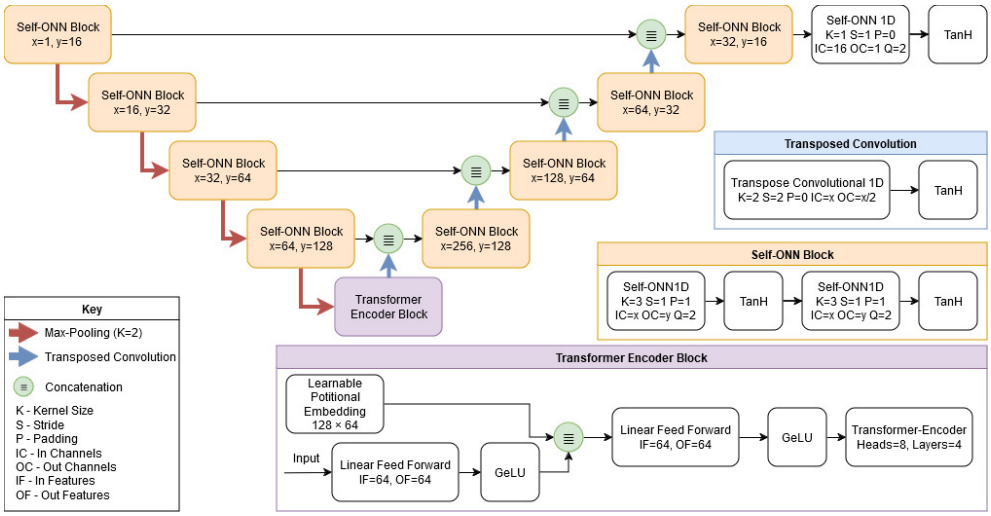



Figure 15: 1D Self-ONN UNet with a transformer encoder bottleneck and 2 Self-ONN layers per encoder and decoder

function. Following testing, it was found that the use of batch normalisation led to overall worse results. Therefore, it was removed from the network.

As the Self-ONN layers were expected to provide better results due to their non-linear nature, more the network was changed again. The network was reverted to the convolutional variant, except this time the convolutional layers were directly replaced with Self-ONN layers. These Self-ONN layers were given a Q-value of 2 and the network was tested again. An overview of the network architecture can be seen in figure 15. The theory was that this should provide a fair comparison as the Self-ONN layers directly replaced the convolutional layers. This network also failed to outperform the standard convolutional results.

The original Self-ONN variant of the network was tested again, excluding batch normalisation, but for 20 epochs. The theory was that the increased complexity of the

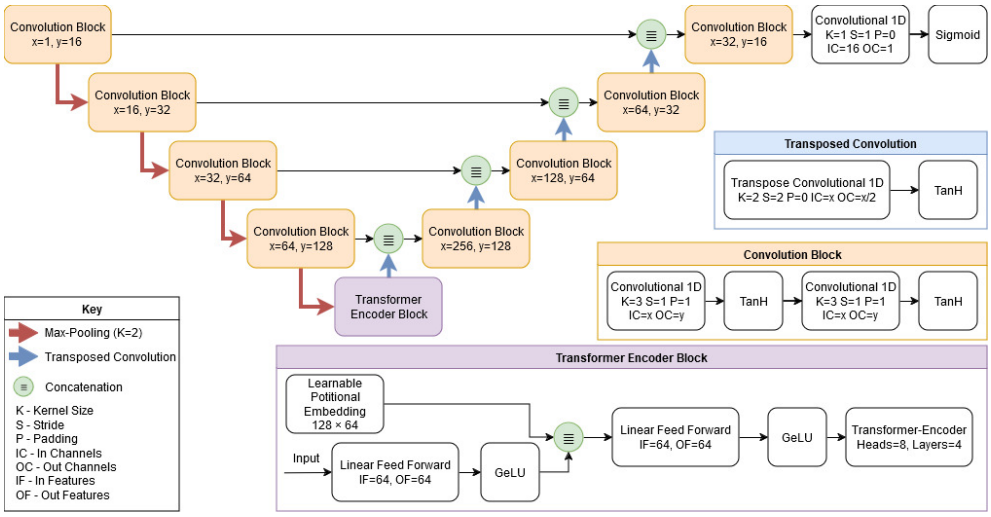



Figure 16: 1D convolutional UNet with a transformer encoder bottleneck and sigmoid for the final activation function

Self-ONN layers may require more training time to be properly effective. This was compared to the convolutional variant of the network which was trained again for the same number of epochs.

The Self-ONN variant once again failed to outperform the convolutional variant even though the convolutional variant contained 194,720 fewer parameters. Therefore, Self-ONN layers were removed from the network and the convolutional layers were reimplemented.

## 6.7 Activation Functions

In an attempt to further improve performance, the activation functions used were brought into question. Due to the use of Tanh activation functions in the network, outputs are in the range of -1 to 1. While this produced encouraging results to this point, the preprocessing of the EEG signals normalised the data to values between 0 and 1. Network output could be renormalised between this range; however, it would likely lead to worse results since the network is not being trained with this final transformation taken into account. To circumvent this issue, it was hypothesised that replacing the final activation function in the network with a Sigmoid activation function would lead to a better overall temporal correlation between the network output and ground truth signals. The network architecture can be found in figure 16.

Following testing, a small improvement was found from the use of the Sigmoid activation function. Therefore, it was kept in the model.

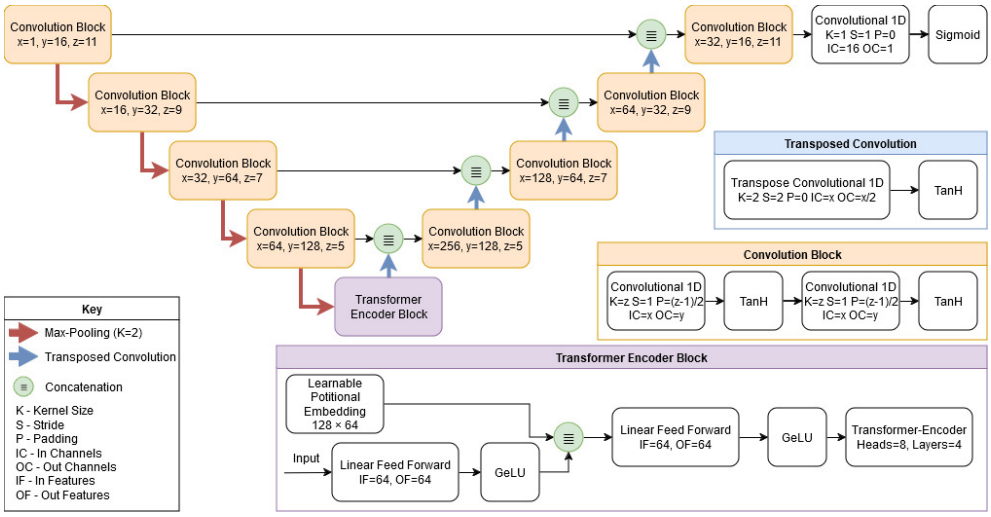



Figure 17: Final model architecture

## 6.8 Kernel Sizes

For further improvement, different kernel sizes were considered throughout the network. [6] suggested the use of larger kernel sizes in higher encoder and decoder layers in the network. This is to allow for the consideration of more information at higher resolutions, leading to the network gaining a better understanding of the underlying distribution of the data. While this will increase the number of parameters in the network, the potential of it leading to better results needed to be explored.

Kernel size was adjusted down the network. The encoder and decoder with the lowest number of channels were given a kernel size of 11. Each layer deeper in the network, the kernel size was reduced by 2, leading to the kernels on the deepest layer, layer 4, having a kernel size of 5. To account for this change in kernel size and to maintain the same output resolution as before, the padding of each kernel was also changed. The padding applied to each kernel was $\frac{k-1}{2}$ where $k$ is the kernel size. All other parameters were maintained for this trial. The network architecture can be seen in figure 17.

An improvement in overall performance was found from the changes made to the kernel sizes through the network. Following comparison with existing methods, it was found that the trained network was achieving competitive results. Therefore, the focus was shifted to the training technique.

## 6.9 CycleGAN

As discussed earlier, the size of the available dataset is limited, even with overlapping windows used to increase the amount of trainable data. One way of solving this was

to use unpaired training so that multiple datasets could be used. This has the benefit of being able to use the existing data and pad it out with datasets containing motion and others which were based on stationary tasks. For example, motor imagery tasks. To test this, a CycleGAN was created for its ability to work with unpaired data and its use in other studies (for example [6]).

Since CycleGANs are made up of 4 different networks, 2 generators and 2 discriminators, iteration was much slower. However, with the generators already designed from the previous process, designing the discriminator network was the next step. PatchGANs are commonly used for this task in both 1D and 2D tasks [75], [76]. As the job of the discriminator is to differentiate between real and fake inputs, it is important that the discriminator is able to provide accurate and specific feedback for the generators. The advantage of a PatchGAN is that it gives feedback on sections of the generator output rather than the whole output. This means that it learns to predict which portions of the generator output seem real and which seem generated, allowing the training process to focus more specifically on significant areas.

The designed PatchGAN discriminator was based on the one proposed by [6]. Due to the previously discovered poor performance of the Self-ONN layers, however, they were replaced with double convolutional layers. This creates a lighter weight network enabling faster testing. Also, the final convolutional layer of the PatchGAN was followed by a sigmoid activation function to allow for a value of 1 to be used for real values and 0 to be used for fake. Although it would have been possible to use a value of -1 for fake signals, to make full use of the Tanh activation function, it limits training in a few ways. For example, Binary Cross-Entropy (BCE) loss expects inputs between 0 and 1 so using Tanh would limit choices for loss functions.

The CycleGAN was developed from scratch using built-in functions in PyTorch. While developing the CycleGAN, the same dataset as before was used to ensure that the inclusion of additional data did not obscure the results. This also meant that previous results could give a rough idea of what to expect which helped with the iteration process.

CycleGANs take a long time to train. This is due to the training of 4 different networks and having to calculate 8 loss values across the networks. As such, development had to be iterative and largely based on intuition. This meant monitoring training was required to be able to quickly improve on results. CycleGANs are complex and difficult to balance, so optimising the performance is much more of an art than a science.

The most important hyper-parameters in a CycleGAN are: the cycle consistency loss multiplier, the identity loss multiplier and the learning rate for the optimisers. As all of these factors are directly related to the loss functions, the loss functions had to be selected first.

To begin the development process, standard hyper-parameter values for CycleGANs were implemented. As in the original CycleGAN [47], a cycle consistency loss multiplier of 10 and an identity loss multiplier of 5 were used. For learning rate, the $5 \times 10^{-5}$ was used as in [6]. However, after quick iteration based on visual inspection, the cycle consistency loss multiplier was reduced to 8 and the identity loss multiplier reduced to 4. This was because the adversarial loss was found to be unstable and the reduction of these multipliers led to better results. It was also found that the discriminators would often output values of around 0.5 from the very first epoch. As a result, and to improve training speeds, the discriminator learning rate was increased to $1 \times 10^{-4}$.

Initially, the optimiser selected for use in the network was Adam. This was because of the success found using Adam previously. However, it was theorised that the use of Root-Mean-Square (RMS) propagation might be more useful due to its ability to work with sparse gradients. This was found to be the case and the optimiser for both the discriminator and generator were replaced. The reason for this improvement is that CycleGANs often require large datatsets to train effectively. As the dataset does not meet this threshold, it is probable that the use of RMS propagation improved results due to Adam struggling to generalise on the lack of data.

Multiple loss functions for cycle consistency loss, identity loss, adversarial loss and discriminator loss were trialled. These included BCE, MAE and MSE. Following this, it was decided the BCE would be used for both adversarial and discriminator loss and MSE would be used for cycle consistency and identity loss as these loss functions were found to be the most effective.

Finally, to help the discriminators better learn to differentiate between real and fake inputs over time, pooling was added. At the start of each batch, the motion contaminated data and ground truth data are both passed through their respective generators. The output of this is then sent to the respective pool. For each output sent to the pool, there is a 50% chance that it will be discarded. Otherwise, the output is added to the pool. If the pool is full, the oldest item in the pool is removed to make space for a new one. In this case, the pool size was set to 32 (double the batch size of 16). Then, when the discriminator is trained to differentiate between real and generated data each epoch, the generated data is randomly sampled from the pool to

ensure the discriminator learns the features of motion contamination more effectively and does not forget previously learned features.

Though time had gone into optimising the training process, consistently worse results than using back-propagation were found. This is to be expected, as using an indirect training technique on matched data will obfuscate the training process. However, the comparison of these results to those containing additional training data was still a useful metric. Therefore, additional datasets were included in the training.

Finding additional data proved challenging. Very few datasets that contain motion contamination are available. One was found in which healthy and epileptic participants performed activities of daily living while wearing an EEG headset with a single electrode [77]. These activities include: walking, standing and cooking. For compatibility with the original dataset, only healthy records were used. Each recording in the dataset is 20 seconds long and there are only 60 recording of healthy patients. As a result the dataset is incredibly limited. Nevertheless, as the only option available, it was added to the training data.

Recordings in this dataset were taken at 512 Hz. Therefore, the downsampling step during preprocessing was only by a factor of 2 rather than 8. As there was no ground truth comparison signal, the window overlap was set to a fixed 50% as in [22] since dynamically adjusting the overlap based on temporal correlation was not an option. Other than these steps, all preprocessing steps matched that applied previously. This resulted in 480 additional samples for training that would be used in all training folds.

To supplement this additional data, a dataset containing resting EEG data was added [78]. This was selected because the participants in the original study [17] were at rest. The dataset was sampled at 2,048 Hz but is already downsampled to 256 Hz. This matches the downsampled frequency of the rest of the datasets used, so it could be directly implemented without downsampling. Only the data featuring open eyes was used and the signal from the electrode in position Fp1 was selected to line up with the position of the electrodes used in [17]. Window overlap was set to a fixed amount, 244 samples or around 23.8%, in order to create a matching number of samples as the additional motion contaminated data.

Following this, the CycleGAN was trained across the 23 folds using the same parameters as before to ensure a fair comparison. The inclusion of this additional data was found to provide worse overall results. Therefore, the network produced from standard back-propagation methods was selected as the final network.

Using back-propagation, a preprocessing technique and neural network were

developed which can outperform existing real-time techniques at removing motion contamination from EEG signals. This is accomplished by creating a dataset with a larger focus on high levels of motion contamination. As a result, fewer network parameters are required making the process less computationally expensive. This demonstrates an improvement to existing processes overall. However, the use of a CycleGAN was found to inhibit results both with and without additional data. This could be due to several factors including the quality of the available datasets and the tuning of the CycleGAN hyper-parameters.

# 7 Results

The aim of this work was to produce a neural network and a preprocessing method to remove motion contamination from EEG signals. To do this, a publicly available dataset, "Motion Artifact Contaminated fNIRS and EEG Data" from [17], was used.

Claims from other research [18], stated that some trials should be excluded. Namely, trials 12 and 15 were stated to not contain any brain data and are poorly correlated. To assess this claim, and the rest of the dataset, the correlation of the signals of each participant was assessed. For each trial, the data was first. The detrended signals were then range normalised between 0 and 1. The results can be found in table 3.

From these results, it is clear that trial 12 in particular has a high correlation compared to the average PCC. In fact, both trials 12 and 15 have above average correlations for the dataset. Figure 7 shows both trial 12 and trial 15 following linear detrending and range normalisation between 0 and 1. It is apparent from these that the ground truth and motion contaminated signals contain relevant and useful information. For example, all 4 periods of artificial motion contamination in each of the motion contaminated signals are clearly visible and appear to have no direct impact on the ground truth signal from a visual inspection.

To further support this claim, a visual inspection of the other trials was also conducted for comparison. When compared with other trials, for example trials 1 and 2 as seen in figure 6, it seems that trials 12 and 15 should, in fact, be included in the training data, contrary to beliefs of [18].

However, from the results in table 3, low correlations were found for trials 10 and 11. Trial 10 in particular contains a noticeable artifact in the ground truth signal (see figure 18). Also of note are trials 20 and 21 for the same reason (also in figure 18).

The removal of these sections of the trials was considered. However, it was

| Trial | Pearson Correlation Coefficient |
|---|---|
| 1 | 0.46142 |
| 2 | 0.24333 |
| 3 | 0.14948 |
| 4 | 0.21095 |
| 5 | 0.92668 |
| 6 | 0.08925 |
| 7 | 0.12998 |
| 8 | 0.48080 |
| 9 | 0.11601 |
| 10 | -0.04659 |
| 11 | -0.05109 |
| 12 | 0.82712 |
| 13 | 0.28500 |
| 14 | 0.16721 |
| 15 | 0.36637 |
| 16 | 0.43930 |
| 17 | 0.43100 |
| 18 | 0.76188 |
| 19 | 0.75491 |
| 20 | 0.24918 |
| 21 | 0.22086 |
| 22 | 0.09085 |
| 23 | 0.44084 |
| Average | 0.33673 |

Table 3: Ground Truth and Motion Contaminated Signal Correlation by Trial

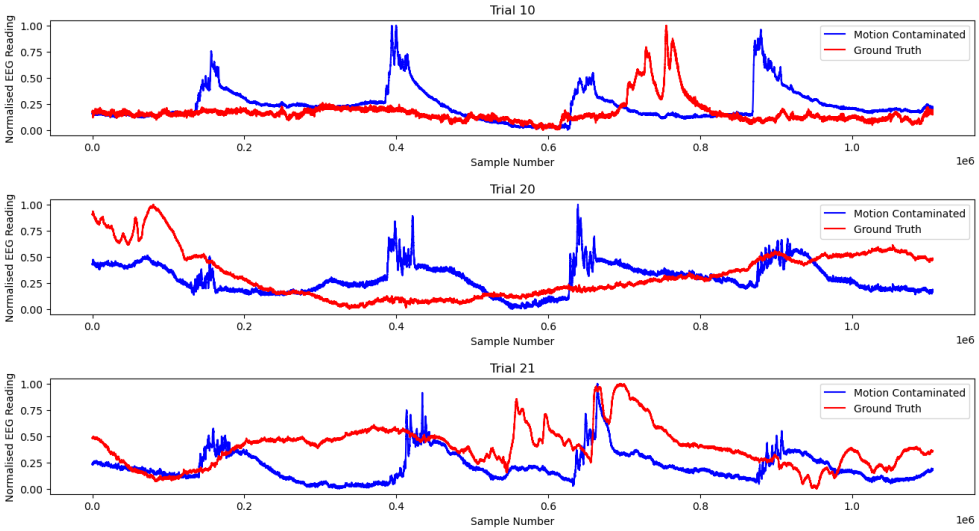



Figure 18: Normalised ground truth and motion contaminated signals from trials 10, 20 and 21

| Trial | PCC |
|---|---|
| 1 | 0.93737 |
| 2 | 0.78579 |
| 3 | 0.81510 |
| 4 | 0.77402 |
| 5 | 0.73693 |
| 6 | 0.68348 |
| 7 | 0.73015 |
| 8 | 0.79368 |
| 9 | 0.79553 |
| 10 | 0.75843 |
| 11 | 0.83766 |
| 12 | 0.91828 |
| 13 | 0.70710 |
| 14 | 0.69438 |
| 15 | 0.76627 |
| 16 | 0.76791 |
| 17 | 0.87307 |
| 18 | 0.87891 |
| 19 | 0.87632 |
| 20 | 0.86919 |
| 21 | 0.89030 |
| 22 | 0.84025 |
| 23 | 0.81158 |
| Average | 0.80616 |
| Standard Deviation | 0.07002 |

Table 4: Temporal correlation per trial for the [22] dataset augmentation method

ultimately decided that the artefactual signals of these trials should remain a part of the data. Predominantly, this decision was made to follow the existing studies [6], [22], which do not mention excluding these segments. As a means of assessing the network architecture accurately against these other methods, trials 10, 20 and 21 were used in their entirety.

## 7.1 Dataset Augmentations

In order to accurately assess the data preprocessing methods from the other studies [6], [22], the proposed methods were used to create variations of the dataset.

The preprocessing and windowing method described in [22] resulted in an augmented dataset with 6187 samples. The dataset features an average temporal correlation of 0.80616 between samples (see table 4) with a ratio of 1842:4345 for contaminated to non-contaminated segments (based on a PCC of 0.5).

The result of the steps described in [6] is a dataset with 3259 samples and an average PCC of 0.45468 between samples (see table 5) with a ratio of 903:2356 for contaminated to non-contaminated segments (based on a PCC of 0.5). To supplement this for validation, an additional copy of this augmentation was created with a 50%

| Trial | PCC |
|---|---|
| 1 | 0.61591 |
| 2 | 0.44198 |
| 3 | 0.52472 |
| 4 | 0.37297 |
| 5 | 0.32924 |
| 6 | 0.26913 |
| 7 | 0.28632 |
| 8 | 0.46626 |
| 9 | 0.44198 |
| 10 | 0.37412 |
| 11 | 0.46279 |
| 12 | 0.64747 |
| 13 | 0.39141 |
| 14 | 0.38697 |
| 15 | 0.46307 |
| 16 | 0.43108 |
| 17 | 0.54802 |
| 18 | 0.52212 |
| 19 | 0.41215 |
| 20 | 0.52814 |
| 21 | 0.55416 |
| 22 | 0.45702 |
| 23 | 0.51148 |
| Average | 0.45385 |
| Standard Deviation | 0.09396 |

Table 5: Temporal correlation per trial for the [6] dataset augmentation method

overlap rather than a dynamic one for comparison with the previous augmentation.

One thing of note is how much of either dataset is made up of severe motion contamination (a PCC below 0.5). 29.8% and 27.7% respectively. As a result, the networks are largely being trained to reproduce the input with a smaller focus on motion contamination removal. The proposed dataset aimed to increase the ratio of severe motion contaminated segments using a different window overlapping equation (see equation 8).

For the custom dataset, a polynomial with an order of 10 was initially used for the fit to detrend both the ground truth and motion contaminated signals. Resulting from this is an augmented dataset with 4050 samples and an average PCC of 0.79394 between samples (see table 6) with a ratio of 1748:2302 for contaminated to non-contaminated segments (based on a PCC of 0.5). The percentage of samples that contain a significant amount of motion contamination, a PCC of over 0.5, is 43.16% which is a large increase over the other 2 data augmentation methods, both of which have values below 30%.

However, in later trials, it was found that the network failed to improve the temporal correlation on some of the trials. This suggested that the order of the

| Trial | PCC |
|---|---|
| 1 | 0.94406 |
| 2 | 0.83940 |
| 3 | 0.80893 |
| 4 | 0.74312 |
| 5 | 0.74022 |
| 6 | 0.75332 |
| 7 | 0.72871 |
| 8 | 0.74431 |
| 9 | 0.78375 |
| 10 | 0.75605 |
| 11 | 0.84026 |
| 12 | 0.92250 |
| 13 | 0.64223 |
| 14 | 0.72411 |
| 15 | 0.67581 |
| 16 | 0.68031 |
| 17 | 0.86974 |
| 18 | 0.90157 |
| 19 | 0.81416 |
| 20 | 0.80591 |
| 21 | 0.88555 |
| 22 | 0.86812 |
| 23 | 0.78840 |
| Average | 0.79394 |
| Standard Deviation | 0.07952 |

Table 6: Temporal correlation per trial for the proposed dataset augmentation method with a polynomial of order 10 to detrend motion contaminated signal

| Trial | PCC |
|---|---|
| 1 | 0.86349 |
| 2 | 0.73339 |
| 3 | 0.73840 |
| 4 | 0.55030 |
| 5 | 0.58657 |
| 6 | 0.58765 |
| 7 | 0.56652 |
| 8 | 0.64777 |
| 9 | 0.68674 |
| 10 | 0.65559 |
| 11 | 0.74774 |
| 12 | 0.84939 |
| 13 | 0.53723 |
| 14 | 0.62726 |
| 15 | 0.53552 |
| 16 | 0.57279 |
| 17 | 0.74983 |
| 18 | 0.75650 |
| 19 | 0.66910 |
| 20 | 0.69796 |
| 21 | 0.76330 |
| 22 | 0.76071 |
| 23 | 0.72264 |
| Average | 0.67854 |
| Standard Deviation | 0.09413 |

Table 7: Temporal correlation per trial for the proposed dataset augmentation method with a polynomial of order 4 to detrend motion contaminated signal

polynomial being applied to the motion contaminated signal was too strong to the point that it was removing important information from the signal. Therefore, the order of the polynomial used to fit the motion contaminated signals was reduced to 4.

The final custom data preparation method featured a temporal correlation of 0.67854 between the ground truth and motion contaminated segments. All other factors of the dataset remained the same. A breakdown of the correlations of the final method can be found in table 7.

For a clearer comparison, the temporal correlation of the 3 datasets can be seen in table 8.

The table clearly displays a number of key points about the proposed method. Firstly, it sits between the other methods in terms of the average correlation of the windows, with an average correlation of 0.67854 as opposed to 0.80616 and 0.45385. This displays an improvement over [6] in terms of preprocessing accuracy. Detrending the motion contaminated signal allows the network to learn more when it comes to motion contamination removal and less about general detrending from baseline drift.

Also visible in the table is the closer correlation between the proposed method

| Trial | [22] Segments | [6] Segments | Proposed Segments |
|---|---|---|---|
| 1 | 0.93737 | 0.61591 | 0.86349 |
| 2 | 0.78579 | 0.44198 | 0.73339 |
| 3 | 0.81510 | 0.52472 | 0.73840 |
| 4 | 0.77402 | 0.37297 | 0.55030 |
| 5 | 0.73693 | 0.32924 | 0.58657 |
| 6 | 0.68348 | 0.26913 | 0.58765 |
| 7 | 0.73015 | 0.28632 | 0.56652 |
| 8 | 0.79368 | 0.46626 | 0.64777 |
| 9 | 0.79553 | 0.44198 | 0.68674 |
| 10 | 0.75843 | 0.37412 | 0.65559 |
| 11 | 0.83766 | 0.46279 | 0.74774 |
| 12 | 0.91828 | 0.64747 | 0.84939 |
| 13 | 0.70710 | 0.39141 | 0.53723 |
| 14 | 0.69438 | 0.38697 | 0.62726 |
| 15 | 0.76627 | 0.46307 | 0.53552 |
| 16 | 0.76791 | 0.43108 | 0.57279 |
| 17 | 0.87307 | 0.54802 | 0.74983 |
| 18 | 0.87891 | 0.52212 | 0.75650 |
| 19 | 0.87632 | 0.41215 | 0.66910 |
| 20 | 0.86919 | 0.52814 | 0.69796 |
| 21 | 0.89030 | 0.55416 | 0.76330 |
| 22 | 0.84025 | 0.45702 | 0.76071 |
| 23 | 0.81158 | 0.51148 | 0.72264 |
| Average | 0.80616 | 0.45385 | 0.67854 |
| Standard Deviation | 0.07002 | 0.09396 | 0.09413 |

Table 8: Data augmentation and segmentation comparison

and the [22] method as opposed to the correlation between the[6] method and the [22] method. As the preprocessing steps in the proposed method and the method proposed by [6] are practical for real-time use, this displays that the preprocessing on the data allows the network to focus more on the removal of motion contamination artifacts.

Finally, this table also displays the increase in motion artifacts in the dataset by having the highest standard deviation of the approaches. Despite the proposed approach having more preprocessing than the approach of [6], it still has a higher standard deviation displaying the increase in the amount of segments containing significant motion contamination.

## 7.2 Dataset Evaluation

To evaluate the quality of the proposed dataset preparation methods, they were all used to train an identical network. A 1D UNet was designed for this purpose (see figure 11).

To form fair assessments, the method presented in [6] was supplemented with an additional validation set. The validation set used the same windowing method as [22] with a 50% overlap to ensure fair and accurate comparison. This additional validation

| Fold | Dataset | Network Processed |
|---|---|---|
| 1 | 0.93737 | 0.96718 |
| 2 | 0.78579 | 0.89570 |
| 3 | 0.81510 | 0.92082 |
| 4 | 0.77402 | 0.79917 |
| 5 | 0.73693 | 0.80471 |
| 6 | 0.68348 | 0.77253 |
| 7 | 0.73015 | 0.78900 |
| 8 | 0.79368 | 0.90092 |
| 9 | 0.79553 | 0.90184 |
| 10 | 0.75843 | 0.87697 |
| 11 | 0.83766 | 0.91315 |
| 12 | 0.91828 | 0.96670 |
| 13 | 0.70710 | 0.82553 |
| 14 | 0.69438 | 0.82933 |
| 15 | 0.76627 | 0.86529 |
| 16 | 0.76791 | 0.86721 |
| 17 | 0.87307 | 0.94382 |
| 18 | 0.87891 | 0.96010 |
| 19 | 0.87632 | 0.96377 |
| 20 | 0.86919 | 0.94983 |
| 21 | 0.89030 | 0.96896 |
| 22 | 0.84025 | 0.94597 |
| 23 | 0.81158 | 0.91406 |
| Average | 0.80616 | 0.89315 |
| Standard Deviation | 0.07002 | 0.06215 |

Table 9: PCC for each fold of the [22] dataset using a 1D UNet before and after processing with a 1D UNet

set was also created for the proposed preprocessing method for the same reason.

Each dataset was trained for 10 epochs with a learning rate of $2 \times 10^{-4}$ using the Adam optimiser (with betas of 0.9 and 0.999) and the MSE loss function. This training was completed with the leave-one-out approach, so each dataset was trained 23 times. For the proposed data preparation method and the method from [6], the trial used for validation in each fold was substituted with the validation set featuring a 50% overlap between windows as in [22].

First, the [22] dataset variation was trained. The results for each fold can be found in table 9. It displays results spanning a range of 0.11, with all PCC results between the values of 0.77253 and 0.96896. Originally, after preprocessing, this version of the dataset had an average PCC of 0.80616. By increasing the correlation to 0.89315, on average, the network shows an improvement of around 10.798%. However, the network performed particularly poorly on folds 4, 6 and 7. Nevertheless, these results still show a noticeable improvement following training which is likely due to heavy preprocessing to detrend the data.

In order to assess the quality of these results, the same test was repeated. However,

| Fold | Validation Dataset | Network Processed |
|---|---|---|
| 1 | 0.62576 | 0.93844 |
| 2 | 0.43891 | 0.87048 |
| 3 | 0.52981 | 0.88947 |
| 4 | 0.37639 | 0.71579 |
| 5 | 0.32120 | 0.76380 |
| 6 | 0.27636 | 0.73701 |
| 7 | 0.29970 | 0.72478 |
| 8 | 0.46390 | 0.86485 |
| 9 | 0.44584 | 0.85365 |
| 10 | 0.37478 | 0.80152 |
| 11 | 0.46119 | 0.85934 |
| 12 | 0.66600 | 0.94411 |
| 13 | 0.38083 | 0.71369 |
| 14 | 0.40098 | 0.75282 |
| 15 | 0.45707 | 0.78352 |
| 16 | 0.45629 | 0.79290 |
| 17 | 0.55055 | 0.91080 |
| 18 | 0.52624 | 0.92837 |
| 19 | 0.41358 | 0.93165 |
| 20 | 0.53018 | 0.92331 |
| 21 | 0.57372 | 0.94036 |
| 22 | 0.45634 | 0.91159 |
| 23 | 0.53141 | 0.89259 |
| Average | 0.45900 | 0.84543 |
| Standard Deviation | 0.09687 | 0.07970 |

Table 10: PCC for each fold of the [6] dataset with 50% window overlap before and after processing with a 1D UNet

this time using the dataset created following the steps of [6]. Theoretically, this should achieve worse results than the previous method. Several factors contribute to this assumption. The first is the lack of training data. There are 2,928 fewer records in this dataset meaning the dataset is around 47.325% smaller. Having roughly half the amount of data should theoretically produce worse results. In addition, the dataset features a slightly worse ratio of motion contaminated data to data which does not contain contamination (27.7% as opposed to 29.8% containing a PCC below 0.5) making it more difficult for the network to learn the appropriate features and identifiers to look out for.

Table 10 shows the results of the second dataset augmentation technique. As hypothesised, the results are significantly lower with an average PCC following processing of 0.84543, a decrease of 0.04772. That being said, there is a larger increase in temporal correlation based on the use of the network. This can be attributed to a number of factors including the task of general detrending being less challenging than the process of motion artifact removal as well as the less processed dataset having more room for correlational improvement. Nevertheless, the overall PCC was lower

than when using the previous technique.

There was also a larger spread of results with a standard deviation of 0.0797 across the 23 folds. This result falls short of beating the previous method in consistency as well, with the previous achieving 0.06215, missing out by 0.01755. This margin is relatively large. However, it is worth noting that this dataset could be used in a practical scenario due to its consistent preprocessing technique which is not dependent upon the correlation between the ground truth and motion contaminated signals.

The results shown were evaluated using the validation version of the dataset created, making these results directly comparable to the previous results. The validation set does not include the dynamic overlap technique used for the creation of the dataset used in [6] but rather a 50% overlap. However, the rest of the processing remains the same. The worse results are a result of fewer preprocessing steps in comparison with the other augmentation technique, such as strong detrending prior to windowing, as well as the use of significantly less data for the model to train on.

Finally, the network was trained on the proposed dataset augmentations. The expectation for this test was that it would produce better results than the [6] dataset. This is due to applying a detrending operation to the motion contaminated signal as well as placing greater emphasis on the motion contaminated portions of the signal, making up over 43% of the dataset instead of less than 30%. However, due to the lack of strong preprocessing steps prior to windowing, it was theoretically unlikely to outperform the dataset used in [22]. The aim was, to get comparable results with less intensive preprocessing that would be practical in a real-time environment and not have to be applied after the fact. As a result, the theory was that the custom dataset should be slightly out-performed by the dataset used in [22].

As expected, it achieved a lower PCC than the preprocessing method used by [22], with an average PCC of 0.85610 compared to 0.89315. This can be attributed to the dataset being only 65.46% the size of the dataset used in [22] and having less specific preprocessing steps. However, it performed better than the preprocessing method of [6] by a PCC of 0.01067. This performance can be attributed to the detrending applied to the motion contaminated signal as well as greater emphasis on motion contaminated segments due to the window overlapping technique used in the creation of the training dataset.

In addition, the proposed preprocessing method sits in the middle of all 3 methods when it comes to standard deviation, showing an increase in consistency over the method proposed by [6]. This shows the proposed dataset to be a reasonable middle

| Fold | Validation Dataset | Network Processed |
|---|---|---|
| 1 | 0.88097 | 0.94435 |
| 2 | 0.75705 | 0.88347 |
| 3 | 0.80039 | 0.89746 |
| 4 | 0.61865 | 0.73193 |
| 5 | 0.64501 | 0.76337 |
| 6 | 0.62193 | 0.74947 |
| 7 | 0.63200 | 0.74645 |
| 8 | 0.75073 | 0.87362 |
| 9 | 0.74115 | 0.86496 |
| 10 | 0.71566 | 0.81552 |
| 11 | 0.77407 | 0.87116 |
| 12 | 0.87525 | 0.94658 |
| 13 | 0.63782 | 0.73978 |
| 14 | 0.66884 | 0.78015 |
| 15 | 0.67034 | 0.79982 |
| 16 | 0.68606 | 0.80289 |
| 17 | 0.81712 | 0.91572 |
| 18 | 0.81022 | 0.92635 |
| 19 | 0.80862 | 0.93942 |
| 20 | 0.80640 | 0.92468 |
| 21 | 0.82878 | 0.94547 |
| 22 | 0.79275 | 0.92729 |
| 23 | 0.78279 | 0.90033 |
| Average | 0.74446 | 0.85610 |
| Standard Deviation | 0.08039 | 0.07476 |

Table 11: PCC for each fold of the adjusted proposed dataset with 50% window overlap before and after processing with a 1D UNet

| Fold | [22] Dataset | [6] Dataset | Proposed Dataset |
|---|---|---|---|
| 1 | 0.96718 | 0.93844 | 0.94435 |
| 2 | 0.89570 | 0.87048 | 0.88347 |
| 3 | 0.92082 | 0.88947 | 0.89746 |
| 4 | 0.79917 | 0.71579 | 0.73193 |
| 5 | 0.80471 | 0.76380 | 0.76337 |
| 6 | 0.77253 | 0.73701 | 0.74947 |
| 7 | 0.78900 | 0.72478 | 0.74645 |
| 8 | 0.90092 | 0.86485 | 0.87362 |
| 9 | 0.90184 | 0.85365 | 0.86496 |
| 10 | 0.87697 | 0.80152 | 0.81552 |
| 11 | 0.91315 | 0.85934 | 0.87116 |
| 12 | 0.96670 | 0.94411 | 0.94658 |
| 13 | 0.82553 | 0.71369 | 0.73978 |
| 14 | 0.82933 | 0.75282 | 0.78015 |
| 15 | 0.86529 | 0.78352 | 0.79982 |
| 16 | 0.86721 | 0.79290 | 0.80289 |
| 17 | 0.94382 | 0.91080 | 0.91572 |
| 18 | 0.96010 | 0.92837 | 0.92635 |
| 19 | 0.96377 | 0.93165 | 0.93942 |
| 20 | 0.94983 | 0.92331 | 0.92468 |
| 21 | 0.96896 | 0.94036 | 0.94547 |
| 22 | 0.94597 | 0.91159 | 0.92729 |
| 23 | 0.91406 | 0.89259 | 0.90033 |
| Average | 0.89315 | 0.84543 | 0.85610 |
| Standard Deviation | 0.06215 | 0.07970 | 0.07476 |

Table 12: All dataset results for a 1D UNet

ground between the 2 previous preprocessing methods.

It is again worth noting that the dataset used for validation of the network, trained on the custom dataset, featured the same preprocessing as the training set. However, the overlap of the windows was set to 50%, as in [22], to make the results directly comparable.

The training time for this network varied across the different datasets and directly reflects the size of the data created by each of the windowing methods. The proposed windowing method required 97 seconds per fold, the [22] dataset required 152 seconds per fold and the [6] dataset required 86 seconds per fold. An inference time of 0.0028 seconds was recorded for this network.

For easier comparison, table 12 displays the results from all 3 trials. Following the claims of [18] that trials 12 and 15 should be excluded from the dataset, it is worth noting that there is no significant difference in performance in these trials. In fact, in all dataset variations, fold 12 performed above average. Trial 15, on the other hand, resulted in below average results on all dataset variations. With that being said, it was far from obtaining the lowest PCC in any of the trials. For that reason, these trials were kept in the dataset.

| Fold | [22] Overlap | [6] Overlap | Proposed Overlap |
|---|---|---|---|
| 1 | 0.94435 | 0.94388 | 0.94058 |
| 2 | 0.88347 | 0.88294 | 0.87571 |
| 3 | 0.89746 | 0.89040 | 0.87387 |
| 4 | 0.73193 | 0.73006 | 0.67223 |
| 5 | 0.76337 | 0.75956 | 0.72883 |
| 6 | 0.74947 | 0.74461 | 0.73387 |
| 7 | 0.74645 | 0.73714 | 0.69680 |
| 8 | 0.87362 | 0.87142 | 0.84428 |
| 9 | 0.86496 | 0.86098 | 0.85805 |
| 10 | 0.81552 | 0.81747 | 0.78369 |
| 11 | 0.87116 | 0.87142 | 0.86414 |
| 12 | 0.94658 | 0.94389 | 0.94083 |
| 13 | 0.73978 | 0.74369 | 0.68201 |
| 14 | 0.78015 | 0.75786 | 0.75851 |
| 15 | 0.79982 | 0.80325 | 0.76224 |
| 16 | 0.80289 | 0.79287 | 0.75118 |
| 17 | 0.91572 | 0.91717 | 0.90001 |
| 18 | 0.92635 | 0.92281 | 0.91410 |
| 19 | 0.93942 | 0.94067 | 0.92313 |
| 20 | 0.92468 | 0.92579 | 0.90557 |
| 21 | 0.94547 | 0.94303 | 0.93444 |
| 22 | 0.92729 | 0.92618 | 0.91703 |
| 23 | 0.90033 | 0.89824 | 0.88828 |
| Average | 0.85610 | 0.85328 | 0.83258 |
| Standard Deviation | 0.07476 | 0.07645 | 0.08867 |

Table 13: Proposed dataset validated using leave-one-out with different window overlaps on the validation sets

Also mentioned before were trials 10, 20 and 21 for regions of contamination in the ground truth signal. Trial 10 achieved results below the average PCC on all processing methods. However, although it was below average, it was not on the lower end of the results for any of the datasets. Furthermore, trials which were not identified as containing motion contamination in the ground truth signal, for example trial 4, perform even more poorly across all dataset variations. Therefore, the exclusion of this trial cannot be justified.

In addition, trials 20 and 21 both score above the average PCC on every variation of the dataset. This indicates that the impact of the identified contamination is negligible. As a result, the exclusion of these trials is also unnecessary.

For further evaluation, the 1D UNet trained on the proposed dataset was validated against itself using the leave-one-out approach. This time, the dataset used for the evaluation of each fold featured the same dynamic overlap used for the training set. As a result, the test set contained more motion contamination for testing against. Also, both [22] and [6] used this approach for validation purposes as standard in the leave-one-out approach. Table 13 shows the result of this trial.

Using this approach to validation, the network is tested against a larger amount of motion contaminated data. With this knowledge, it is understood that worse results should be expected when testing with the leave-one-out approach. This testing was done to use as a benchmark for progress throughout the development of the network. From the initial results, it can be seen that the use of a trained UNet improves the PCC of the dataset by 5.4% from 0.78758 to 0.83011. Although it is an improvement over the preprocessing alone, it shows the requirement of a more complex network to better identify motion contamination and remove it from the data.

For completeness, the proposed preprocessing method was applied to the data but with the window overlap used by [6]. This was done to produce a value that could be directly compared with the results stated in the paper. [6] claims a temporal correlation of 0.872 following preprocessing of the signal and processing by the network. Table 13 shows a temporal correlation of 0.85946 with the proposed preprocessing technique and equivalent window overlap to [6]. This displays that, even with a standard 1D UNet, the proposed preprocessing method can train a simple network of comparable performance, with a discrepancy in the leave-one-out validation value of only 0.021.

## 7.3 Base Architecture Selection

In order to identify a base architecture to proceed with, the previously mentioned UNet was tested against an FPN with a similar number of parameters. The test was completed using the same hyper-parameters as before but on the 2 different networks. In addition, the proposed dataset preparation steps were used for both training and validation.

It was hypothesised that the UNet should significantly outperform the FPN for this task due to its additional processing steps in the bottleneck and the greater number of channels in the decoder path. Ideally, the FPN would perform similarly to the UNet, allowing for a lighter weight solution to the problem. The designed FPN is shown in figure 12.

The FPN had an average training time per fold of 95 seconds with an inference time of 0.0026 seconds and results of these trials can be found in table 14. From these results, a number of observations can be drawn. It is notable that the FPN performs better than the UNet on fold 3. This displays the potential of the FPN architecture for this task being more light-weight and computationally inexpensive compared to the UNet.

| Fold | FPN | UNet |
|---|---|---|
| 1 | 0.93356 | 0.94140 |
| 2 | 0.86735 | 0.86940 |
| 3 | 0.89622 | 0.87768 |
| 4 | 0.63853 | 0.67477 |
| 5 | 0.69907 | 0.71954 |
| 6 | 0.72232 | 0.72832 |
| 7 | 0.67773 | 0.69192 |
| 8 | 0.80276 | 0.82802 |
| 9 | 0.85313 | 0.85805 |
| 10 | 0.76893 | 0.78616 |
| 11 | 0.84891 | 0.86434 |
| 12 | 0.92700 | 0.94231 |
| 13 | 0.67267 | 0.68201 |
| 14 | 0.75837 | 0.75864 |
| 15 | 0.74536 | 0.76125 |
| 16 | 0.70822 | 0.75118 |
| 17 | 0.88251 | 0.90001 |
| 18 | 0.91207 | 0.91302 |
| 19 | 0.89717 | 0.91934 |
| 20 | 0.89346 | 0.90382 |
| 21 | 0.93236 | 0.93688 |
| 22 | 0.90298 | 0.91666 |
| 23 | 0.88321 | 0.88828 |
| Average | 0.81843 | 0.83100 |
| Standard Deviation | 0.09448 | 0.08942 |

Table 14: PCC comparison FPN vs UNet

However, it is evident that on the majority of folds, the performance of the network underperforms that of the UNet. Whilst some other trials provide comparable results, such as trials 2, 6, and 9, it is was clear that the network suffered from the lack of a bottleneck and the concatenation of the encoder outputs to the decoder inputs. With an average PCC across the folds of 0.81843 and a standard deviation of 0.09448, the FPN was also found to be less consistent than the UNet, making the decision to use a UNet based architecture a simple one.

## 7.4 Transformer Encoder Bottleneck

Following the previous trial displaying that the UNet outperformed the FPN, the UNet was modified. Special attention was given to the bottleneck which was indicated to be a key component of the UNet's performance.

Inspired by other research [39], the bottleneck convolutions of the UNet were replaced with a transformer encoder. This also featured learned positional encoding which was concatenated with the data prior to processing by the transformer encoder. The inclusion of the transformer encoder increased the training time to an average of 128 seconds per fold with an average inference time of 0.0052 seconds.

| Fold | Convolutional | Transformer |
|---|---|---|
| 1 | 0.94140 | 0.95418 |
| 2 | 0.86940 | 0.88526 |
| 3 | 0.87768 | 0.90342 |
| 4 | 0.67477 | 0.72996 |
| 5 | 0.71954 | 0.73689 |
| 6 | 0.72832 | 0.75234 |
| 7 | 0.69192 | 0.73519 |
| 8 | 0.82802 | 0.82984 |
| 9 | 0.85805 | 0.85827 |
| 10 | 0.78616 | 0.81318 |
| 11 | 0.86434 | 0.88649 |
| 12 | 0.94231 | 0.94925 |
| 13 | 0.68201 | 0.68716 |
| 14 | 0.75864 | 0.76713 |
| 15 | 0.76125 | 0.76331 |
| 16 | 0.75118 | 0.78754 |
| 17 | 0.90001 | 0.91729 |
| 18 | 0.91302 | 0.93455 |
| 19 | 0.91934 | 0.92624 |
| 20 | 0.90382 | 0.90571 |
| 21 | 0.93688 | 0.94852 |
| 22 | 0.91666 | 0.92114 |
| 23 | 0.88828 | 0.89614 |
| Average | 0.83100 | 0.84735 |
| Standard Deviation | 0.08942 | 0.08341 |

Table 15: Comparison of PCC between UNet bottleneck configurations

The new network was trained and validated against the custom dataset. These trials used the same hyper-parameters as before and used leave-one-out validation. The results of these trials can be found in table 15.

The results show an improvement in the PCC of 0.01635 following the inclusion of a transformer in the bottleneck. This is due to its ability to find relationships between different patterns in the signal in a retrospective manner. This increase in accuracy also came with a decrease in standard deviation of the results across the folds from 0.08942 to 0.08341. This displays the usefulness of introducing a transformer encoder with learned positional encoding to the bottleneck as it lead to an improvement in temporal correlation and overall consistency.

## 7.5 Self-ONN Layers

Convolutional layers were removed and replaced with Self-ONN layers due to their use in [6]. The research claimed an increase in performance from the use of Self-ONN layers. To trial this, the double convolutional layers used in the encoders and decoders were replaced with a Self-ONN layer followed by the same activation function as with the convolutional variant.

| Fold | Convolution | Self-ONN | Self-ONN + Batch Normalisation |
|---|---|---|---|
| 1 | 0.95418 | 0.94593 | 0.90827 |
| 2 | 0.88526 | 0.86877 | 0.81733 |
| 3 | 0.90342 | 0.88004 | 0.88140 |
| 4 | 0.72996 | 0.72314 | 0.71108 |
| 5 | 0.73689 | 0.72183 | 0.62920 |
| 6 | 0.75234 | 0.70888 | 0.74120 |
| 7 | 0.73519 | 0.73703 | 0.69848 |
| 8 | 0.82984 | 0.81523 | 0.79185 |
| 9 | 0.85827 | 0.85086 | 0.84516 |
| 10 | 0.81318 | 0.79212 | 0.79816 |
| 11 | 0.88649 | 0.86023 | 0.84537 |
| 12 | 0.94925 | 0.93812 | 0.93691 |
| 13 | 0.68716 | 0.68617 | 0.68944 |
| 14 | 0.76713 | 0.78323 | 0.77457 |
| 15 | 0.76331 | 0.71487 | 0.75920 |
| 16 | 0.78754 | 0.74800 | 0.73141 |
| 17 | 0.91729 | 0.90371 | 0.90041 |
| 18 | 0.93455 | 0.92700 | 0.92766 |
| 19 | 0.92624 | 0.91694 | 0.91271 |
| 20 | 0.90571 | 0.91249 | 0.88475 |
| 21 | 0.94852 | 0.92590 | 0.92591 |
| 22 | 0.92114 | 0.90178 | 0.90078 |
| 23 | 0.89614 | 0.88762 | 0.86454 |
| Average | 0.84735 | 0.83260 | 0.82069 |
| Standard Deviation | 0.08341 | 0.08550 | 0.08803 |

Table 16: A comparison of PCC per fold between the convolutional variant, Self-ONN variant and Self-ONN with batch normalisation variant of the network

As part of this trial, the network was trained once with batch normalisation and once without. The results of this trial can be found in table 16. The average training time per fold for the Self-ONN variant was 109 seconds with an inference time of 0.0054 seconds. For the Self-ONN with batch normalisation variant, the average training time per fold was 108 seconds with an inference time of 0.0059 seconds.

Surprisingly, the results show a decrease in performance when using Self-ONN layers and a further decrease in performance when using batch normalisation between the Self-ONN layers and the activation function. Not only was the average performance of these networks worse but they also featured a wider range of results: shown by a higher standard deviation. These results were confusing given the success found in the work of [6].

In addition, there was an increase in the number of parameters across the network due to the use of the Self-ONN layers (with a Q-value of 3). The standard convolutional network had a parameter count of 1,522,801, whereas the Self-ONN version of the network had a parameter count of 1,717,521. Intuitively, this does not make much sense as a network with a higher parameter counts would be expected to be more

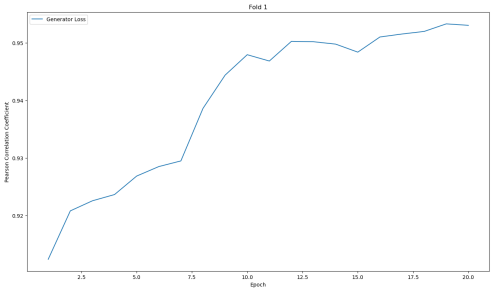
| Fold | Convolution | Self-ONN |
|---|---|---|
| 1 | 0.95418 | 0.94883 |
| 2 | 0.88526 | 0.86174 |
| 3 | 0.90342 | 0.90312 |
| 4 | 0.72996 | 0.70311 |
| 5 | 0.73689 | 0.74359 |
| 6 | 0.75234 | 0.75528 |
| 7 | 0.73519 | 0.72141 |
| 8 | 0.82984 | 0.82692 |
| 9 | 0.85827 | 0.83059 |
| 10 | 0.81318 | 0.80292 |
| 11 | 0.88649 | 0.87991 |
| 12 | 0.94925 | 0.94665 |
| 13 | 0.68716 | 0.68147 |
| 14 | 0.76713 | 0.75968 |
| 15 | 0.76331 | 0.76535 |
| 16 | 0.78754 | 0.76170 |
| 17 | 0.91729 | 0.89338 |
| 18 | 0.93455 | 0.92002 |
| 19 | 0.92624 | 0.91713 |
| 20 | 0.90571 | 0.90717 |
| 21 | 0.94852 | 0.93748 |
| 22 | 0.92114 | 0.88305 |
| 23 | 0.89614 | 0.88908 |
| Average | 0.84735 | 0.83650 |
| Standard Deviation | 0.08341 | 0.08253 |

Table 17: Comparison of PCC per fold between the convolutional variant and the Self-ONN variant of the network, with 2 Self-ONN layers per encoder and decoder with a q value of 2

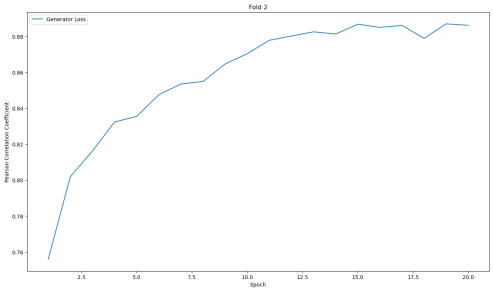
effective.

To better understand the problem, the encoders and decoders of the Self-ONN UNet were changed to more closely reflect those of the convolutional variant. The encoders and decoders featured 2 Self-ONN layers, each followed by a Tanh activation function. To account for the significant increase in parameters, and to give a closer reference to the convolutional UNet, the Q-value of all Self-ONN layers in the network was changed to 2. Batch normalisation was excluded for its poor results in the previous test. The result was a network with 1,816,241 parameters. The result of this test can be found in table 17.

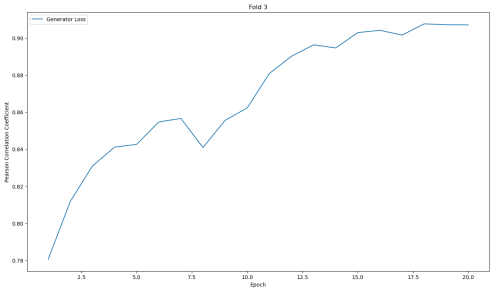
As evidenced by these results, the network still underperformed the standard convolutional UNet, even though it contained 293,440 more parameters. This indicated an issue with the training process as equivalent results would at least be expected. Due to the increase in complexity provided by Self-ONN layers, it was hypothesised that the number of epochs for efficient optimisation of the network was too low. While not the case for the convolutional variant of the network, it was the only reasonable conclusion which could be drawn at this point.




(a) Fold 1




(b) Fold 2




(c) Fold 3

Figure 19: PCC over 20 training epochs on the Self-ONN variant of the network

| | Convolutional | | Self-ONN | |
|---|---|---|---|---|
| | 10 Epochs | 20 Epochs | 10 Epochs | 20 Epochs |
| 1 | 0.95418 | 0.95351 | 0.94883 | 0.95306 |
| 2 | 0.88526 | 0.89374 | 0.86174 | 0.88658 |
| 3 | 0.90342 | 0.91140 | 0.90312 | 0.90758 |
| 4 | 0.72996 | 0.70505 | 0.70311 | 0.73141 |
| 5 | 0.73689 | 0.75054 | 0.74359 | 0.75124 |
| 6 | 0.75234 | 0.77172 | 0.75528 | 0.77002 |
| 7 | 0.73519 | 0.75097 | 0.72141 | 0.74578 |
| 8 | 0.82984 | 0.83775 | 0.82692 | 0.84635 |
| 9 | 0.85827 | 0.87107 | 0.83059 | 0.87270 |
| 10 | 0.81318 | 0.81454 | 0.80292 | 0.80914 |
| 11 | 0.88649 | 0.88755 | 0.87991 | 0.88350 |
| 12 | 0.94925 | 0.95085 | 0.94665 | 0.94973 |
| 13 | 0.68716 | 0.72196 | 0.68147 | 0.70720 |
| 14 | 0.76713 | 0.78243 | 0.75968 | 0.78549 |
| 15 | 0.76331 | 0.80447 | 0.76535 | 0.79374 |
| 16 | 0.78754 | 0.78471 | 0.76170 | 0.79558 |
| 17 | 0.91729 | 0.91918 | 0.89338 | 0.91238 |
| 18 | 0.93455 | 0.94305 | 0.92002 | 0.93630 |
| 19 | 0.92624 | 0.92206 | 0.91713 | 0.93216 |
| 20 | 0.90571 | 0.92394 | 0.90717 | 0.92536 |
| 21 | 0.94852 | 0.95459 | 0.93748 | 0.94751 |
| 22 | 0.92114 | 0.92930 | 0.88305 | 0.92655 |
| 23 | 0.89614 | 0.89816 | 0.88908 | 0.89667 |
| Average | 0.84735 | 0.85576 | 0.83650 | 0.85504 |
| Standard Deviation | 0.08341 | 0.07991 | 0.08253 | 0.07830 |

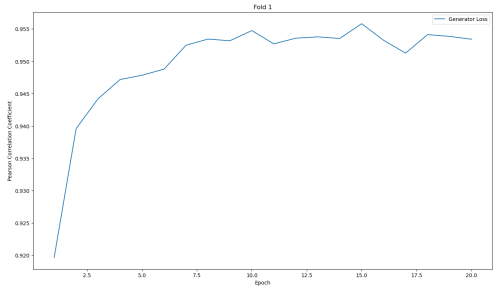
Table 18: Result of 20 epochs on Self-ONN UNet

To test this theory, the PCC of the network was tracked through training. The 1,717,521 parameter variant of the Self-ONN network was used for this test. The number of epochs was increased to 20 to test the hypothesis. The PCC graph throughout this test for folds 1, 2 and 3 can be seen in figure 19.

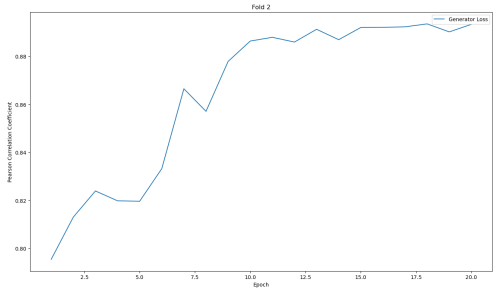
When training for 20 epochs, the network with Self-ONN layers took an average of 205 seconds per fold for training. Additionally, the convolutional variant required an average of 235 seconds per fold.

While the PCC of some of the networks plateaued around 10 epochs, some found greater improvement over the 20 epochs rather than 10 used previously for training. As a result of this improvement in performance, a full trial was run against the dataset. In addition to this, to ensure that the improvement was not the result of improvements to the transformer, the convolutional variant was also trained for 20 epochs as a comparison. The results of this trial can be found in table 18.

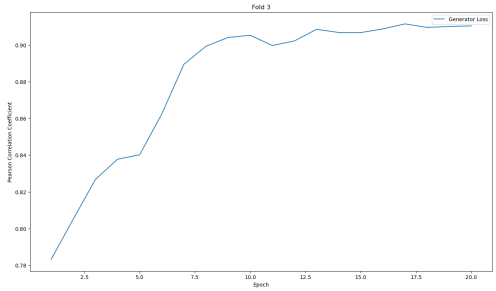
As shown in table 18 the results are an improvement over those in table 16 but at the cost of more parameters than the convolutional variant. It also shows that an increase in the epochs for the convolutional variant offers better results than its Self-ONN




(a) Fold 1




(b) Fold 2




(c) Fold 3

Figure 20: PCC over 20 training epochs on the convolutional variant of the network

counterpart. Although the final result between the 2 variants was ultimately close, the convolutional network was providing similar results with hundreds of thousands fewer parameters.

It was posited that increasing the training epochs further may still result in better results. To assess this, the PCC was assessed over the epochs to gain an understanding of how much improvement could be found. Figure 19 shows the PCC over 20 epochs for the first 3 folds of the Self-ONN network variant. Improvement clearly plateaus before the 20 epochs are complete, showing that the Self-ONN variant was not offering notable improvement over the standard convolutional model. For that reason, the convolutional variant of the network was worked with from here on out as it was proving to perform better with fewer parameters.

Similarly to the Self-ONN variant of the network, figure 20 shows that the training of the convolutional variant of the network has plateaued by 20 epochs of training. However, it is noticeable that in a few of the folds, the best result is not achieved in

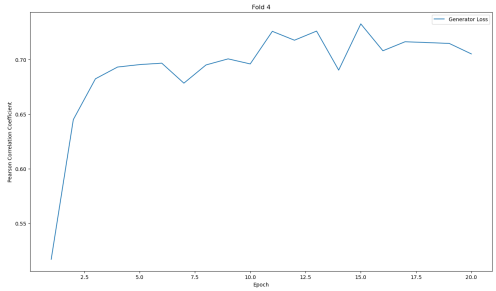



Figure 21: PCC for the convolutional network in fold 4

the last epoch. This is particularly visible in fold 4, shown in figure 21.

It can be seen that the best result is achieved in epoch 15 and that progress is relatively unstable after the 6th epoch. To improve this and to help the network perform better, it would be possible to implement a learning scheduler to decrease the learning rate over the epochs. This should stabilise training and lead to better results overall across the network. It would also be possible to implement a system which saves only the best performing network across training. While both of these methods offer potential, it was first decided to experiment more with the architecture of the network in order to make previous results directly comparable to the new architectures.

## 7.6 Activation Functions

The final Tanh activation function was replaced with a Sigmoid function to ensure the network output was between 0 and 1. This was done as the ground truth data that the network was learning to imitate was also normalised between these values.

To test this hypothesis, the final activation function in the network was switched to Sigmoid. The test was run using the same parameters as in the previous test, except that the number of training epochs was reduced back to 10. While 20 had received better results, it was important to be able to make the architecture changes directly comparable to all previous iterations. It also meant that iteration time could be twice as fast. The comparison in table 19 shows the result of this test compared with the Self-ONN network described for the previous test.

For the network with the final activation function replaced with Sigmoid, the average training time per fold was 126 seconds with an inference time of 0.0054 seconds. The results show a minor improvement in performance from the use of a Sigmoid activation in the final layer with an improvement in PCC of 0.00212. This change also led to an improvement in consistency across the folds. The standard deviation of the results improved from 0.08341 to 0.07820, an improvement of 6.25%.

| Fold | Tanh | Sigmoid |
|---|---|---|
| 1 | 0.95418 | 0.95275 |
| 2 | 0.88526 | 0.86884 |
| 3 | 0.90342 | 0.91042 |
| 4 | 0.72996 | 0.72130 |
| 5 | 0.73689 | 0.75081 |
| 6 | 0.75234 | 0.75590 |
| 7 | 0.73519 | 0.75159 |
| 8 | 0.82984 | 0.83930 |
| 9 | 0.85827 | 0.86889 |
| 10 | 0.81318 | 0.81309 |
| 11 | 0.88649 | 0.88058 |
| 12 | 0.94925 | 0.94986 |
| 13 | 0.68716 | 0.71357 |
| 14 | 0.76713 | 0.78691 |
| 15 | 0.76331 | 0.79358 |
| 16 | 0.78754 | 0.76709 |
| 17 | 0.91729 | 0.91086 |
| 18 | 0.93455 | 0.93627 |
| 19 | 0.92624 | 0.93079 |
| 20 | 0.90571 | 0.87170 |
| 21 | 0.94852 | 0.95289 |
| 22 | 0.92114 | 0.92648 |
| 23 | 0.89614 | 0.88443 |
| Average | 0.84735 | 0.84947 |
| Standard Deviation | 0.08341 | 0.07820 |

Table 19: Final layer activation function results

This further displays that replacing the final Tanh activation function with Sigmoid yielded stronger results.

## 7.7 Optimisers

At the start of the iteration process, the Adam optimiser was somewhat arbitrarily selected for training. Adam is a very commonly used optimiser in machine learning and is widely accepted as a reasonable starting point.

Nevertheless, there are some drawbacks to the use of Adam. For example, it is thought to be poor for generalisation, with some researchers opting to use other, sometimes less precise methods, in search of better generalisation [65].

There are variants of Adam available. One example is AdamW which aims to tackle the issue of poor generalisation by separating weight decay from the gradient update process [79]. By handling these values separately, issues which can arise from them being linked, such as interference from adaptive learning rates, can be avoided resulting in a more stable training process.

Another variant of Adam, Adamax [80], simplifies the gradient scaling process in the optimiser by using the maximum absolute value of the gradients instead of the L2

| Fold | Adam | AdamW | Adamax |
|---|---|---|---|
| 1 | 0.95275 | 0.95235 | 0.94641 |
| 2 | 0.86884 | 0.86849 | 0.86614 |
| 3 | 0.91042 | 0.90938 | 0.85794 |
| 4 | 0.72130 | 0.71788 | 0.69657 |
| 5 | 0.75081 | 0.75020 | 0.72369 |
| 6 | 0.75590 | 0.75315 | 0.73508 |
| 7 | 0.75159 | 0.75406 | 0.73304 |
| 8 | 0.83930 | 0.83902 | 0.80244 |
| 9 | 0.86889 | 0.86881 | 0.85802 |
| 10 | 0.81309 | 0.81358 | 0.80560 |
| 11 | 0.88058 | 0.88133 | 0.86838 |
| 12 | 0.94986 | 0.94970 | 0.92940 |
| 13 | 0.71357 | 0.71376 | 0.67087 |
| 14 | 0.78691 | 0.78677 | 0.76632 |
| 15 | 0.79358 | 0.79336 | 0.77139 |
| 16 | 0.76709 | 0.76678 | 0.73865 |
| 17 | 0.91086 | 0.91078 | 0.90595 |
| 18 | 0.93627 | 0.93678 | 0.92808 |
| 19 | 0.93079 | 0.93056 | 0.89400 |
| 20 | 0.87170 | 0.87123 | 0.84962 |
| 21 | 0.95289 | 0.95185 | 0.91801 |
| 22 | 0.92648 | 0.92646 | 0.91919 |
| 23 | 0.88443 | 0.88646 | 0.84813 |
| Average | 0.84947 | 0.84925 | 0.82752 |
| Standard Deviation | 0.07820 | 0.07843 | 0.08171 |

Table 20: Optimiser comparison for convolutional UNet

normal. This makes it more robust to large gradients.

As there are multiple variants of Adam available, AdamW and Adamax were also experimented with. Using the previous configuration, AdamW and Adamax were also trialled. The result of these trials can be found in table 20.

The AdamW trial had an average training time of 121 seconds per fold and the Adamax optimiser had an average training time of 124 seconds per fold. The results show close output from all of the optimisers. As all of the optimisers are built on the same foundation, this is not surprising. However, it does show that a standard Adam optimiser gives the best performance for this task over the other options. While AdamW offered almost identical performance to that of the plain Adam optimiser, Adamax performed noticeably worse (especially on mean PCC with a difference of over 0.02). This is likely due to Adamax being designed for tasks with large gradients which is typically suited to much larger neural networks targeted at much more complex tasks.

These results proved that a standard Adam optimiser was the correct choice originally.

| Fold | Kernel Size 3 | Staggered Kernel Sizes |
|---|---|---|
| 1 | 0.95275 | 0.95460 |
| 2 | 0.86884 | 0.89084 |
| 3 | 0.91042 | 0.91731 |
| 4 | 0.72130 | 0.74183 |
| 5 | 0.75081 | 0.76887 |
| 6 | 0.75590 | 0.76155 |
| 7 | 0.75159 | 0.76869 |
| 8 | 0.83930 | 0.83551 |
| 9 | 0.86889 | 0.87305 |
| 10 | 0.81309 | 0.82114 |
| 11 | 0.88058 | 0.89228 |
| 12 | 0.94986 | 0.95301 |
| 13 | 0.71357 | 0.72977 |
| 14 | 0.78691 | 0.79384 |
| 15 | 0.79358 | 0.80578 |
| 16 | 0.76709 | 0.77997 |
| 17 | 0.91086 | 0.91970 |
| 18 | 0.93627 | 0.94516 |
| 19 | 0.93079 | 0.92122 |
| 20 | 0.87170 | 0.92774 |
| 21 | 0.95289 | 0.95499 |
| 22 | 0.92648 | 0.92729 |
| 23 | 0.88443 | 0.89304 |
| Average | 0.84947 | 0.85988 |
| Standard Deviation | 0.07820 | 0.07516 |

Table 21: Comparison of using a kernel size of 3 on for all kernels vs using larger kernels at higher layers

## 7.8 Kernel Sizes

For the final adjustment to the network, the kernel sizes and padding were adjusted to account for the resolution of the data at the current level in the network. Therefore, higher-resolution encoders and decoders received larger kernels than those deeper in the network.

This network was tested using the same hyper-parameters as with the previous trial. The results can be seen in table 21.

The new network had an average training time per fold of 131 seconds with an inference time of 0.0053 seconds. As suggested, the increase in kernel size led to a deeper understanding by the network and a further increase in the PCC between the network output and ground truth. Due to this increase, the network output from the folds was saved for testing against the other windowing techniques. However, it was discovered that the network was not behaving as expected once saved and the PCC was more erratic when testing against even the same dataset.

As mentioned previously, it was hypothesised that decreasing the learning rate throughout training would improve results. This was based on the results from previous

| Fold | Previous PCC (10 epochs) | Best PCC | Best Epoch | Final PCC |
|---|---|---|---|---|
| 1 | 0.95460 | 0.95579 | 19 | 0.95530 |
| 2 | 0.89084 | 0.90335 | 17 | 0.89446 |
| 3 | 0.91731 | 0.92035 | 12 | 0.91883 |
| 4 | 0.74183 | 0.74921 | 19 | 0.74226 |
| 5 | 0.76887 | 0.77325 | 19 | 0.76318 |
| 6 | 0.76155 | 0.79260 | 15 | 0.78388 |
| 7 | 0.76869 | 0.77560 | 17 | 0.76852 |
| 8 | 0.83551 | 0.86688 | 16 | 0.84502 |
| 9 | 0.87305 | 0.88145 | 14 | 0.88190 |
| 10 | 0.82114 | 0.82267 | 16 | 0.82270 |
| 11 | 0.89228 | 0.89211 | 16 | 0.89064 |
| 12 | 0.95301 | 0.94912 | 18 | 0.95436 |
| 13 | 0.72977 | 0.72942 | 18 | 0.72396 |
| 14 | 0.79384 | 0.80446 | 14 | 0.80143 |
| 15 | 0.80578 | 0.81320 | 13 | 0.81326 |
| 16 | 0.77997 | 0.81426 | 14 | 0.80913 |
| 17 | 0.91970 | 0.92445 | 17 | 0.92380 |
| 18 | 0.94516 | 0.94474 | 20 | 0.94531 |
| 19 | 0.92122 | 0.94343 | 13 | 0.93527 |
| 20 | 0.92774 | 0.93135 | 19 | 0.92893 |
| 21 | 0.95499 | 0.95640 | 14 | 0.95516 |
| 22 | 0.92729 | 0.93329 | 12 | 0.93087 |
| 23 | 0.89304 | 0.90564 | 11 | 0.90398 |
| Average | 0.85988 | 0.86883 | 16 | 0.86488 |
| Standard Deviation | 0.07516 | 0.07217 | 2.58687 | 0.07431 |

Table 22: PCC results from training the UNet with modified transformer bottleneck over 20 epochs

training in which the PCC would become unstable late in training, as evidenced in figure 21. As a result, a linear learning rate decay was implemented such that the learning rate in the final epoch was 1% of that in the first epoch. In addition, the number of epochs was adjusted back to 20 as this was the final trial intended to be run.

As it was the final trial for this method, each fold was monitored and the highest performing network from the 20 epochs was saved for each trial. This was to ensure maximum accuracy of the network. The results of this trial can be found in table 22.

With the increase to 20 epochs, the mean training time per fold for the final network was around 274 seconds. The results show a clear improvement in overall score from the additional epochs and a decrease in learning rate over time. This is displayed by the comparison of the previous runs (shown in the second column of the table) to the PCC after the 20th epoch of training in each fold (shown in the final column of the table). An increase of the average PCC across the folds, of 0.005, and a decrease in the standard deviation of the PCCs across the folds, of 0.00214, demonstrates an overall increase in both accuracy and consistency.

| Fold | [22] windowing | [6] windowing | Proposed Windowing |
|---|---|---|---|
| 1 | 0.96057 | 0.95813 | 0.95579 |
| 2 | 0.91416 | 0.91265 | 0.90335 |
| 3 | 0.93120 | 0.92588 | 0.92035 |
| 4 | 0.80982 | 0.80578 | 0.74921 |
| 5 | 0.81666 | 0.80742 | 0.77325 |
| 6 | 0.81037 | 0.80285 | 0.79260 |
| 7 | 0.82349 | 0.81268 | 0.77560 |
| 8 | 0.89651 | 0.89567 | 0.86688 |
| 9 | 0.88719 | 0.88560 | 0.88145 |
| 10 | 0.85595 | 0.85556 | 0.82267 |
| 11 | 0.90113 | 0.90156 | 0.89211 |
| 12 | 0.96109 | 0.95843 | 0.94912 |
| 13 | 0.78790 | 0.78731 | 0.72942 |
| 14 | 0.82758 | 0.82930 | 0.80446 |
| 15 | 0.86553 | 0.86750 | 0.81320 |
| 16 | 0.85266 | 0.85581 | 0.81426 |
| 17 | 0.94016 | 0.94299 | 0.92445 |
| 18 | 0.95368 | 0.95218 | 0.94474 |
| 19 | 0.96016 | 0.96244 | 0.94343 |
| 20 | 0.94778 | 0.94799 | 0.93135 |
| 21 | 0.96566 | 0.96412 | 0.95640 |
| 22 | 0.94162 | 0.94064 | 0.93329 |
| 23 | 0.91842 | 0.91508 | 0.90564 |
| Average | 0.89258 | 0.89077 | 0.86883 |
| Standard Deviation | 0.05764 | 0.05892 | 0.07217 |

Table 23: Network validated against different windowing techniques

From the other 2 columns, it can be seen that, by taking the best result from each fold, a further improvement can be found. The results show the best results to occur commonly around the 16th epoch. This indicates that 20 epochs is, on the whole, more than sufficient for the network on this dataset. An improvement of 0.00395 in terms of average PCC can be found and an additional decrease of just over 0.002 for the standard deviation display an improvement in accuracy and consistency over using the result of the 20th epoch.

However, some of the poorer performing folds, such as folds 4 and 5, earned the highest PCC on fold 19. This indicates that they could potentially benefit from additional training epochs. In this case, it was deemed unnecessary due to the results of the other folds being much more favourable.

To be able to accurately assess the performance of the network over that of [22] and [6], the network was also validated using the window overlapping techniques of both mentioned approaches. The preprocessing remained the same but the window overlap was adjusted. The results of this validation can be found in table 23.

From the results, it can be seen that the proposed preprocessing method and network outperform the method proposed by [6] by a PCC of around 0.0011. [6]

| Trial | Preprocessed | Network Output |
|---|---|---|
| 1 | 0.86349 | 0.95579 |
| 2 | 0.73339 | 0.90335 |
| 3 | 0.73840 | 0.92035 |
| 4 | 0.55030 | 0.74921 |
| 5 | 0.58657 | 0.77325 |
| 6 | 0.58765 | 0.79260 |
| 7 | 0.56652 | 0.77560 |
| 8 | 0.64777 | 0.86688 |
| 9 | 0.68674 | 0.88145 |
| 10 | 0.65559 | 0.82267 |
| 11 | 0.74774 | 0.89211 |
| 12 | 0.84939 | 0.94912 |
| 13 | 0.53723 | 0.72942 |
| 14 | 0.62726 | 0.80446 |
| 15 | 0.53552 | 0.81320 |
| 16 | 0.57279 | 0.81426 |
| 17 | 0.74983 | 0.92445 |
| 18 | 0.75650 | 0.94474 |
| 19 | 0.66910 | 0.94343 |
| 20 | 0.69796 | 0.93135 |
| 21 | 0.76330 | 0.95640 |
| 22 | 0.76071 | 0.93329 |
| 23 | 0.72264 | 0.90564 |
| Average | 0.67854 | 0.86883 |
| Standard Deviation | 0.09413 | 0.07217 |

Table 24: Network improvement over preprocessed dataset

claims a PCC of 0.88 using their method which can be seen to be less effective than the proposed method. In addition to this, the proposed model contains 1,779,953 parameters whereas the model proposed by [6] contained "around 1.953 million parameters". This shows a reduction in parameters of around 173,000.

[22] quotes an average temporal correlation of 0.90516 across the 23 folds. While this is higher than the proposed network, this method uses accelerometer data. As the aim of this research was to produce a network which could be used on any EEG headset for a low cost, this was not practical for the proposed network. However, the difference in performance between the 2 methods overall is less than 2% showing that a comparable level of performance can be found using only the contaminated EEG signals and much less harsh preprocessing.

A comparison between the preprocessed dataset and the final network output can be seen in table 24 . The results show in the table display an average increase in correlation of 28.044% over the preprocessing. This is an improvement over the work of [22] which achieved an improvement of 10.937% over the dataset proposed in the same research. While obtaining better results, this was including the use of accelerometer data in the input to the network and the use of preprocessing steps which cannot be

used in real-time.

In addition, while the results of [6] display a much larger improvement, of 91.721%, it does not achieve as accurate results as that of the proposed method (a PCC of 0.880 as opposed to 0.891 when the proposed technique is validated using the same window overlap method as in [6] for direct comparison). It achieves this improvement due to a lack of preprocessing on the motion contaminated signal, relying on the network to perform both detrending and motion contamination removal. The proposed method also achieves this improved performance with around 173,000 fewer model parameters.

Although the improvement of the temporal correlation is small at around 1.1% (a temporal correlation of 89.1% as opposed to 88% of [6]), it is important. When working with EEG signals, it is essential to provide the most accurate results as the signals are often transformed into power bands for evaluation. Errors in the signal can lead to further errors down the line and lead to greater inaccuracies. This network ultimately provides a more representative output with lower computational overheads. The significance of this is that it was achieved with the use of fewer network parameters due to processing improvements.

Another method commonly used to assess the performance of the motion contamination removal techniques from EEG signals is the SNR. This helps to indicate the improvement in signal quality throughout the process. However, in this case, it is more difficult to accurately assess the SNR as the dataset was originally range normalised between 0 and 1.

$$\Delta SNR = 10log_{10}\left(\frac{\sigma_x^2}{\sigma_{e_{after}}^2}\right) - 10log_{10}\left(\frac{\sigma_x^2}{\sigma_{e_{before}}^2}\right) \tag{9}$$

In order to estimate this improvement, the normalised output was adjusted to fit the same range as the original corresponding ground truth segments without any preprocessing. The calculation of the improvement in SNR ( $\Delta SNR$ ) in equation 9.

In this equation, $\sigma_x^2$ is the squared variance of the ground truth, $\sigma_{e_{before}}^2$ is the squared variance of the motion contaminated signal and $\sigma_{e_{after}}^2$ is the squared variance of the motion-contaminated signal after preprocessing and network processing.

To calculate the improvement in SNR, the original data set, without pre-processing or windowing, was downsampled and segmented using the proposed windowing method. The variance of each trial was then taken. The results of this can be seen in table 25.

After substituting these values into the equation, a SNR improvement of +26.896 dB is shown. This is an improvement over the methods of [22] and [6] which achieved improvements of +26.641 dB and +26.497 dB respectively. This displays a modest

| Trial | Ground Truth | Motion Contaminated | Network Processed |
|---|---|---|---|
| 1 | 623.85155 | 1968.70889 | 721.23227 |
| 2 | 5530.22292 | 32030.90834 | 1924.14373 |
| 3 | 637.57896 | 76015.52723 | 747.10467 |
| 4 | 446.20861 | 6816.76931 | 312.44512 |
| 5 | 431.43176 | 7043.61854 | 282.40952 |
| 6 | 547.28432 | 12830.86544 | 313.97586 |
| 7 | 565.51214 | 31233.26774 | 301.75250 |
| 8 | 317.64572 | 11205.90300 | 356.80315 |
| 9 | 1193.09087 | 10052.35460 | 658.02797 |
| 10 | 600.11354 | 11347.92082 | 489.91348 |
| 11 | 573.70937 | 5120.19056 | 531.05701 |
| 12 | 648.43762 | 7893.74150 | 759.60468 |
| 13 | 2076.93146 | 9413.05141 | 1048.57959 |
| 14 | 12489.51028 | 9029.84794 | 4280.50909 |
| 15 | 887.70940 | 17880.64464 | 581.55407 |
| 16 | 643.28949 | 9203.60979 | 460.25886 |
| 17 | 738.13886 | 9010.24316 | 732.20301 |
| 18 | 1148.04933 | 12222.57919 | 1169.20118 |
| 19 | 2194.05098 | 76281.67084 | 1861.15471 |
| 20 | 1050.57687 | 31130.50348 | 1084.70678 |
| 21 | 915.37798 | 34594.10648 | 1149.18950 |
| 22 | 2071.49606 | 22470.90564 | 1252.95632 |
| 23 | 610.44315 | 37254.08534 | 772.79058 |
| Average | 1606.11571 | 20958.74017 | 947.45972 |
| Standard Deviation | 2557.78618 | 19855.88599 | 838.32364 |

Table 25: Variance from proposed windowing

improvement over the other methods of between 0.255 dB and 0.399 dB.

However, it is arguable that these results are not directly comparable again due to the difference in windowing. As the windowing on the proposed method focuses more heavily on the motion contaminated portion of the results, this is likely to create a higher SNR as there will be a greater amount of variance in segments containing significant motion contamination.

To compare with the other results, the same method was repeated. The original, unfiltered ground truth and motion contaminated signals were segmented using the same methods as [22] and [6]. The final networks created from the training process were then also assessed using the variants of the proposed dataset featuring the window overlap methods from the other studies. The results for the [6] variant of the dataset can be found in table 26 and the results for the [22] variant can be found in table 27.

With these results for comparison, an SNR improvement of +22.430 dB can be found for the [6] windowing method and +22.794 dB for the [6] windowing method. Although still a significant improvement, this is below the figures quoted by the other methods by around 4 dB. This drop can be attributed to both a lower average variance of the ground truth signals (over 250 in both cases) and a much lower average of the

| Trial | Ground Truth | Motion Contaminated | Network Processed |
|---|---|---|---|
| 1 | 627.19224 | 1755.46678 | 723.97759 |
| 2 | 4451.10946 | 22229.73602 | 1853.47091 |
| 3 | 617.97098 | 32840.25173 | 743.57470 |
| 4 | 449.98128 | 4073.47692 | 327.78561 |
| 5 | 453.80993 | 6129.68833 | 309.28680 |
| 6 | 495.89706 | 8571.31096 | 312.33133 |
| 7 | 582.76228 | 22164.83436 | 327.84629 |
| 8 | 328.95595 | 5799.56175 | 376.54968 |
| 9 | 1205.38145 | 7133.64481 | 752.88448 |
| 10 | 660.34550 | 9276.72658 | 546.02591 |
| 11 | 575.00721 | 4585.15966 | 534.72670 |
| 12 | 645.08922 | 5866.61665 | 737.28726 |
| 13 | 2375.21778 | 4801.25188 | 1101.19584 |
| 14 | 5905.56673 | 7347.44322 | 2842.50462 |
| 15 | 689.35943 | 6353.78957 | 492.13584 |
| 16 | 548.13410 | 4518.84604 | 423.44374 |
| 17 | 731.43930 | 4962.16749 | 737.11646 |
| 18 | 1038.45127 | 7215.32690 | 1072.01080 |
| 19 | 2538.47513 | 27726.44438 | 2114.38531 |
| 20 | 1001.15404 | 15164.80231 | 1079.75987 |
| 21 | 960.24090 | 19074.66573 | 1121.22074 |
| 22 | 1762.31503 | 18340.05121 | 1150.22247 |
| 23 | 654.62439 | 25179.59642 | 815.71537 |
| Average | 1273.84698 | 11787.42868 | 891.10688 |
| Standard Deviation | 1348.31192 | 8778.46154 | 617.91191 |

Table 26: Variance from [6] windowing

| Trial | Ground Truth | Motion Contaminated | Network Processed |
|---|---|---|---|
| 1 | 636.91575 | 1517.64547 | 728.18815 |
| 2 | 3692.89437 | 30441.57125 | 1722.73705 |
| 3 | 638.97287 | 39243.54929 | 762.92592 |
| 4 | 449.36183 | 3625.02203 | 331.12956 |
| 5 | 466.91709 | 5628.10085 | 308.63663 |
| 6 | 520.33243 | 9509.03219 | 318.10634 |
| 7 | 558.96497 | 19974.98153 | 319.16662 |
| 8 | 328.33939 | 6188.05824 | 373.21674 |
| 9 | 1445.85775 | 6302.15033 | 820.93085 |
| 10 | 657.93523 | 8513.66945 | 533.50345 |
| 11 | 600.27529 | 4397.52302 | 555.41048 |
| 12 | 649.92881 | 4967.62855 | 739.06947 |
| 13 | 2348.81851 | 4779.23790 | 1146.90513 |
| 14 | 8374.55024 | 6072.65307 | 3106.22504 |
| 15 | 681.48589 | 7222.60189 | 499.51799 |
| 16 | 581.35469 | 4505.76237 | 432.35479 |
| 17 | 749.66285 | 4896.58186 | 739.38363 |
| 18 | 1083.92248 | 7494.19194 | 1099.84881 |
| 19 | 2399.69931 | 30051.78663 | 2054.09246 |
| 20 | 1004.57076 | 14426.82080 | 1068.04014 |
| 21 | 958.97153 | 21153.23303 | 1131.43242 |
| 22 | 1658.07044 | 21383.99459 | 1144.76987 |
| 23 | 623.65792 | 23522.64450 | 784.78655 |
| Average | 1352.67219 | 12426.88873 | 900.88600 |
| Standard Deviation | 1691.40633 | 10273.60004 | 642.59365 |

Table 27: Variance from [22] windowing

motion contaminated signals (over 8,000 in both cases). This is in contrast to the change in variance of the network processed output which had a range of less than 57 across all 3 windowing methods.

In addition, the decrease can be attributed to an increase in motion contaminated data used for training over the other methods. This trains the network to have more focus on removing motion contamination instead of focusing more on improving the SNR. In the other methods, over 70% of the training data was focused on matching signals which already had a PCC of over 0.5. This gives greater focus to motion contamination removal and more focus to noise reduction to match the preprocessed signal. The greater focus on motion contamination can be credited as one of the reasons the proposed network achieves a greater PCC than the other methods with fewer parameters.

However, these results demonstrate the consistency of the proposed method due to the low change in output variance between the different windowing techniques.

## 7.9 CycleGAN

Following the successful results using the designed architecture, a CycleGAN was used to train the network. This was to enable the network to be trained on a larger amount of data as the data used for training could be compiled from multiple different sources.

To establish a point of comparison for the additional data, the CycleGAN was first trained on and validated against the proposed dataset using the leave-one-out approach. The results of this training can be found in table 28.

As expected, the training time was significantly increased using the CycleGAN. This is due to the complexity of training multiple networks at once and the requirement of additional epochs. The average training time per fold was 4,889 seconds.

The results show a clear decrease in the temporal correlation of the network output and the ground truth over the previous method (0.79647 as opposed to 0.86883). This is to be expected for multiple reasons.

Firstly, due to time constraints, each fold of the network was only trained for 50 epochs. While this is sufficient to get a relative idea of how effective the CycleGAN is, it is likely that training for a greater number of epochs, with the inclusion of additional fine tuning of the hyper-parameters, would lead to better results.

Secondly, as training took place on a matched dataset, it can be expected that standard back-propagation would result in better results than an indirect training method such as a CycleGAN. As it is the job of the discriminator in a CycleGAN to

| Fold | PCC |
|---|---|
| 1 | 0.93818 |
| 2 | 0.83913 |
| 3 | 0.81927 |
| 4 | 0.69130 |
| 5 | 0.69353 |
| 6 | 0.72628 |
| 7 | 0.69057 |
| 8 | 0.77837 |
| 9 | 0.77773 |
| 10 | 0.75288 |
| 11 | 0.83310 |
| 12 | 0.93016 |
| 13 | 0.65152 |
| 14 | 0.72097 |
| 15 | 0.70555 |
| 16 | 0.69583 |
| 17 | 0.85900 |
| 18 | 0.90023 |
| 19 | 0.84930 |
| 20 | 0.85657 |
| 21 | 0.90406 |
| 22 | 0.87273 |
| 23 | 0.83253 |
| Average | 0.79647 |
| Standard Deviation | 0.08518 |

Table 28: Temporal correlation with CycleGAN with no additional data

identify the key patterns of the target data, which is then passed on to the generator, the important features become obscured due to the additional training step.

However, an increase in the SNR was found. The SNR increased from 26.896 dB in the standard trials to 26.967 dB in the CycleGAN trials. This displays that the CycleGAN may do a better job at identifying the underlying signal than the previous method.

However, it is also possible that the decrease in temporal correlation has led to results with lower variance due to not identifying the motion contamination as accurately. If confusion has been introduced, overcorrection in the signal may improve the SNR whilst producing a worse temporal correlation.

Although the results from the use of a CycleGAN were considerably worse than those of the previous method, additional trials were run to see if an improvement in results could be found from the inclusion of additional datasets.

When searching for other datasets, very few that contain motion could be found. However, using the additional dataset that was found, 480 additional samples were added to the motion contaminated signals. In addition, a dataset containing participants at rest was sampled and added to the dataset to create a matching 480

| Fold | PCC |
|---|---|
| 1 | 0.93050 |
| 2 | 0.82395 |
| 3 | 0.81913 |
| 4 | 0.67710 |
| 5 | 0.68160 |
| 6 | 0.71152 |
| 7 | 0.67776 |
| 8 | 0.75616 |
| 9 | 0.78240 |
| 10 | 0.72773 |
| 11 | 0.81512 |
| 12 | 0.91428 |
| 13 | 0.64369 |
| 14 | 0.70309 |
| 15 | 0.68967 |
| 16 | 0.67686 |
| 17 | 0.84936 |
| 18 | 0.89231 |
| 19 | 0.82966 |
| 20 | 0.83352 |
| 21 | 0.88220 |
| 22 | 0.86175 |
| 23 | 0.81713 |
| Average | 0.78246 |
| Standard Deviation | 0.08560 |

Table 29: Temporal correlation with CycleGAN with additional data

samples added to the ground truth segments used in each trial.

Due to the additional data, the average training time per fold increased to 5,750 seconds. The results of this trial can be found in table 29. From this table, it can be seen that the inclusion of the additional data decreased the final temporal correlation of the results by around 1.4%. While this is not a large difference, it does indicate that the inclusion of this data hampered training overall.

In contrast, a further improvement of the SNR, up to 26.970 dB from 26.967 dB, displays that the inclusion of additional data may have improved the signal quality overall. On the other hand, it is also possible that this is also due to the network failing to identify the motion contamination and is instead producing outputs with missing data leading to an increase in the SNR and a decrease in the temporal PCC.

Originally, the idea of including additional datasets was to improve the training of the network. More data should, in theory, lead to better results. These worse results could be due to several factors. Firstly, the quality of the training results is highly dependent on the quality of the data. Due to the limited availability of additional data, expanding the dataset was very difficult and resulted in only a small number of additional trials. It is probable that the additional motion contaminated data was of

little use to the training and has perhaps caused the worse results. As it was the only other dataset containing motion artifacts found, the available options were incredibly limited. This highlights a requirement for the creation of more motion contaminated datasets in order to achieve better results.

Additionally, the dataset used to supplement the motion contaminated data was collected with the MindWave 2 headset. This is not a traditional EEG headset as it only features the data from a single electrode. In addition, it is an active electrode as opposed to the passive electrode commonly used in EEG research. This difference in electrode type will provide different results due to the requirement of amplification of the signal from active electrodes.

Furthermore, the MindWave 2 is made of a rigid structure, which means that motion is more likely to cause a loss of contact between the wearer and the electrode, as opposed to a standard deformable mesh cap used for EEG research. As a result, motion contamination may appear differently than with a standard deformable cap. However, this is difficult to verify through visual inspection. Both of these factors will have created differences in the data which may have made it more difficult for the network to generalise.

This also raises a larger issue in this domain as ultimately the results of using the [17] dataset are only strictly applicable to the headset used in their method and similar headsets. Motion artifacts are created through the motion of wires and electrodes. Depending on how the electrodes and wires are secured and attached, this is likely to look different in each headset. A custom solution is required on a per-headset basis in order to attain optimal results.

In addition, the dataset created by [17] does not include indicators of motion in the brain signals. This is due to participants remaining still whilst artificial motion contamination was induced. Research has investigated how brain signals change during motor imagery [81]. This is when a participant imagines making a movement but does not actually complete the motion. From this, it can be concluded that there is probably important information missing from the [17] dataset which may improve the performance of neural networks when it comes to removing motion artifacts. It is possible that the dataset is not ideal as a result. However, the solution is a compromise as a method of creating a matched dataset for EEG signals would be very difficult to accomplish any other way.

Overall, the best results came from the use of back-propagation on the transformer encoder infused convolutional UNet. This achieved a temporal correlation between the

ground truth and network output of 0.86883 when validated against its own dataset using the leave-one-out-approach. When validated using the same preprocessing steps but with a different window overlapping technique, it outperforms the existing solution for real-time motion contamination removal techniques by 1.08% with around 173,000 fewer parameters.

This is significant in terms of the overhead costs required to run such a network. It means that the network can be run on more cost-effective hardware as it is less computationally expensive, with an inference time of 0.0053 seconds on the aforementioned hardware. This means that it has a lower barrier to entry for implementation in resource-restricted environments such as a clinical setting.

# 8 Conclusion

This research aimed to create a processing pipeline capable of removing motion contamination from EEG signals as effectively and efficiently as possible. The network produced contains fewer than 1.8 million parameters and achieves better results in terms of temporal correlation improvement than other methods.

The first research question aimed to assess how effective the existing preprocessing techniques used on the available EEG data are in other studies. From the identified studies [6], [22] it was found that the existing methods could be improved. One of the identified methods [22] applied preprocessing steps to the data which was impractical for real-time use by detrending across the whole dataset prior to windowing the data. Not having the option of running effectively in real-time is less than ideal, especially in a clinical setting where instant feedback may be crucial. The outlined preprocessing steps would also work differently on different lengths of recordings making the preprocessing technique inconsistent.

In addition, the method described in [6] does not include any preprocessing on the motion contaminated signal. This left room for improvement as detrending EEG signals is standard practice to remove artifacts such as baseline drift. Giving the network the task of also detrending the signal complicates the process for the network, meaning it has multiple features to identify and remove.

The proposed method was shown to be more effective than [6] after using both pre-processing techniques to train the same network. A correlational improvement of 2.07% was found on a 1D convolutional UNet from the changes applied. This is significant as it displays that a simple improvement in the dataset augmentations can

improve techniques without the requirement of more complex networks. In turn, this reduces costs and lowers the barriers to entry for implementing the solution.

The proposed preprocessing method also performed better due to its increased emphasis on motion contamination. The proposed windowing method creates a dataset of which over 43% is made up of highly motion contaminated data (a PCC between the ground truth and motion contaminated signal of less than 0.5). This is a noticeable improvement over the method of [6] in which less than 28% of the segments contained highly motion contaminated data. From this, the network was able to better adapt to the task of removing motion contamination rather than being simply encouraged to reproduce the input. Again, this is beneficial as it means a network with fewer parameters, thus lower overhead costs, can be trained to complete the task. This leads to a more accessible solution.

Another aim of the research was to produce a compact and efficient network that offers at least similar performance to existing methods. The proposed network outperforms the method of [6] by 1.21% (a PCC of around 0.891 as opposed to 0.880 when tested against the same windows of the dataset) with around 9.86% fewer parameters. This demonstrates an improved performance over the previous method with a decrease in parameters. As a result, the final network is less computationally expensive meaning that it can run on cheaper hardware making it more accessible.

Although the proposed method does not quite match that of [22] in terms of temporal correlation, it does work with a method that could be used in real-time applications with less harsh preprocessing. Although the parameter count of the network produced in the research is not explicitly stated, it can be assumed to be significantly higher due to the use of a similar structure with a significant increase of complexity in the encoders and decoders due to the combination of multiple complex techniques. Such a decrease in parameters leads to lower computational complexity which, in turn, leads to better accessibility.

The question of whether the use of recent techniques, such as the inclusion of a transformer encoder in the bottleneck of the network, would improve performance was raised before trials began. A transformer encoder was included in the bottleneck with learned positional encoding concatenated to the input. A noticeable improvement of around 1.6% was found from the inclusion of a 4 layered transformer encoder with 8 heads. This demonstrates the usefulness of a network architecture that can take sequential relationships into account when working with temporal data.

Although good results have been achieved from this work, the issue of comparison

with other methods has been made evident throughout. The issue of there being no standard method for validating neural networks against the "Motion Artifact Contaminated fNIRS and EEG Data" dataset creates issues of objective comparison between approaches. As the leave-one-out method has been used to assess networks but with different window overlaps, the ratio of the dataset windows containing significant motion contamination to those containing data highly correlated with the ground truth signal fluctuates between research methods. This makes a direct comparison of the quoted results futile as no true conclusions can be drawn without additional work.

It is also worth noting that [6], [22] and the proposed method offer different network architectures and preprocessing steps but all achieve a final temporal correlation between the ground truth and network output within 3% of each other. Additionally this is while not achieving more than 91% accuracy in terms of temporal correlation. This could be tied to imperfections in the core dataset as it offers closely correlated but not identical signals based on the electrodes (for both ground truth and motion contamination) being placed close together but not reading from exactly the same position. As such, if a network was to produce 99-100% correlation between the signals, it would likely not be fully representative of the true signal and would likely introduce its own artifacts.

Either way, these similar results with varying techniques allude towards the solution to motion contamination in EEG signals not being solvable purely through algorithms. Instead, the combination of a mechanical solution paired with software would produce the best results. This would be best developed without the use of the dataset from [17] but with custom, simulated data to achieve optimal performance.

Additionally, for the use of unpaired training techniques, and for better results overall, it is essential that more datasets are created. The very limited availability of motion artifact contaminated EEG datasets was highly restrictive in development. Available datasets suffer from a low quantity of data, low numbers of participants and differing headsets making the use of all the data together difficult for a network to generalise. Creating more datasets is essential for creating better solutions to the issue of motion contamination.

In conclusion, the developed processing pipeline for motion contaminated EEG signals, including the preprocessing steps and neural network, offers state-of-the-art results with fewer parameters than alternate methods. It achieves a temporal correlation of over 89% between ground truth and network output (when validated with 50% window overlapping on the [17] dataset), as well as an improvement in

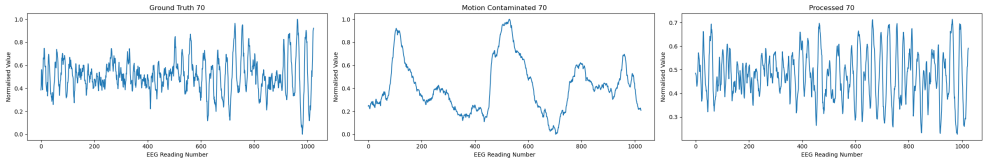



Figure 22: Ground truth, preprocessed motion contamination and network processed example from fold 23 window 70

SNR of over 22 dB (when validated with the same window overlapping technique), demonstrating a significant improvement in signal quality. A demonstration of the improvement provided by the network can be seen in figure 22.

This solution is outperforms similar techniques from [6] whilst reducing the number of parameters. This is significant in reducing the challenges to implementing the solution in resource-restricted environments such as a clinical setting. As the target of this work was to produce a network which could remove motion contamination from EEG signals for use in such a setting, this is a really important accomplishment.

However, further progress is limited by the lack of available data and limitations of existing datasets. To progress further, a hybrid approach that incorporates software and mechanical solutions would likely yield the best results.

# 9 Future Work

Going forward, a standard method for validation, such as the use of 50% overlap, must be maintained for results to be directly comparable. Additional complexity for comparison could also be added through changes to different factors. For example, the use of different window lengths has the potential to impact the comparability of performance. Therefore, a method of accurately assessing machine learning techniques for motion artifact removal from EEG signals is essential.

The development of a headset which is designed to decrease motion artifacts would also prove to be beneficial overall. Any headset designed should aim to reduce the noise introduced by cable and electrode motion. It may be useful in this case to incorporate the machine learning techniques outlined here. However, it would be beneficial to create custom data to train such a model which can be tailored to the specific headset in such a scenario.

Finally, to further progress the work, larger and higher quality datasets are required. Datasets with EEG signals containing motion are difficult to find and much more data is required to effectively train a model. If a large, motion contaminated dataset were to be produced, it would facilitate the use of unpaired training techniques and could

lead to better results overall. This dataset could then be validated using the dataset created by [17].

# References


[1] World Stroke Organization, *World Stroke Organization (WSO): Global Stroke Fact Sheet 2022*, 2022. [Online]. Available: http://ghdx.healthdata.org/gbd-results-tool.

[2] Y. Huang, Y. Li, H. Pan, and L. Han, "Global, Regional, and National Burden of Neurological Disorders in 204 Countries and Territories Worldwide," *Journal of Global Health*, vol. 13, p. 04 160, Nov. 2023, ISSN: 20472986. DOI: 10.7189/jogh.13.04160.

[3] Care Quality Commission, *The State of Health Care and Adult Social Care in England 2022/23*, London, UK, 2023. [Online]. Available: https://www.gov.uk/government/publications/state-of-health-and-adult-social-care-in-england-2022-to-2023.

[4] E. E. Sanches, E. Aupers, N. Sakran, J. Navalta, T. Kostka, and S. Pouwels, "Barriers and Facilitators in Rehabilitation in Chronic Diseases and after Surgery: Is It a Matter of Adherence?" en, *Cureus*, vol. 13, no. 12, e20173, Dec. 2021.

[5] R. Essery, A. W. A. Geraghty, S. Kirby, and L. Yardley, "Predictors of Adherence to Home-Based Physical Therapies: A Systematic Review," *Disability and Rehabilitation*, vol. 39, no. 6, pp. 519–534, 2017. DOI: 10.3109/09638288.2016.1153160.

[6] S. Mahmud, M. E. Chowdhury, S. Kiranyaz, *et al.*, "Restoration of Motion-Corrupted EEG Signals Using Attention-Guided Operational CycleGAN," *Engineering Applications of Artificial Intelligence*, vol. 128, p. 107 514, Feb. 2024, ISSN: 0952-1976. DOI: 10.1016/J.ENGAPPAI.2023.107514.

[7] J. S. Kumar and P. Bhuvaneswari, "Analysis of Electroencephalography (EEG) Signals and Its Categorization–A Study," *Procedia Engineering*, vol. 38, pp. 2525–2536, 2012, International Conference on Modelling Optimization and Computing, ISSN: 1877-7058. DOI: https://doi.org/10.1016/j.proeng.2012.06.298.

[8] E. R. Kande, J. D. Koester, S. H. Mack, and S. A. Siegelbaum, *Principles of Neural Science*. McGraw-Hill Education, 2021, ISBN: 9781259642241. [Online]. Available: www.mhprofessional.com.

[9] G.tec, *G.tec Medical Engineering — Home*. [Online]. Available: https://www.gtec.at/.

[10] A. de Cheveigné and D. Arzounian, "Robust detrending, rereferencing, outlier detection, and inpainting for multichannel data," *NeuroImage*, vol. 172, pp. 903–912, 2018, ISSN: 1053-8119. DOI: https://doi.org/10.1016/j.neuroimage.2018.01.035. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1053811918300351.

[11] A. Widmann, E. Schröger, and B. Maess, "Digital Filter Design for Electrophysiological Data – a Practical Approach," *Journal of Neuroscience Methods*, vol. 250, pp. 34–46, 2015, ISSN: 0165-0270. DOI: https://doi.org/10.1016/j.jneumeth.2014.08.002.

[12] V. Vimala, K. Ramar, and M. Ettappan, "An Intelligent Sleep Apnea Classification System Based on EEG Signals," *Journal of Medical Systems*, vol. 43, no. 2, p. 36, Jan. 2019, ISSN: 1573-689X. DOI: 10.1007/s10916-018-1146-8.

[13] G. Bartur, K. Joubran, S. Peleg-Shani, J.-J. Vatine, and G. Shahaf, "An EEG Tool for Monitoring Patient Engagement during Stroke Rehabilitation: A Feasibility Study," en, *Biomed Research International*, vol. 2017, p. 9 071 568, Sep. 2017.

[14] J. K. Nuamah, Y. Seong, and S. Yi, "Electroencephalography (EEG) Classification of Cognitive Tasks Based on Task Engagement Index," in *2017 IEEE Conference on Cognitive and Computational Aspects of Situation Management (CogSIMA)*, 2017, pp. 1–6. DOI: 10.1109/COGSIMA.2017.7929581.

[15] A. Apicella, P. Arpaia, M. Frosolone, G. Improta, N. Moccaldi, and A. Pollastro, "EEG-Based Measurement System for Monitoring Student Engagement in Learning 4.0," *Scientific Reports*, vol. 12, no. 1, p. 5857, Apr. 2022.

[16] A. de Cheveigné and D. Arzounian, "Robust Detrending, Rereferencing, Outlier Detection, and Inpainting for Multichannel Data," *NeuroImage*, vol. 172, pp. 903–912, May 2018, ISSN: 1053-8119. DOI: 10.1016/J.NEUROIMAGE.2018.01.035.

[17] K. T. Sweeney, H. Ayaz, T. E. Ward, M. Izzetoglu, S. F. McLoone, and B. Onaral, "A Methodology for Validating Artifact Removal Techniques for Physiological Signals," *IEEE Transactions on Information Technology in Biomedicine*, vol. 16, no. 5, pp. 918–926, 2012. DOI: 10.1109/TITB.2012.2207400.

[18] A. K. Maddirala and R. A. Shaik, "Motion Artifact Removal from Single Channel Electroencephalogram Signals Using Singular Spectrum Analysis," *Biomedical Signal Processing and Control*, vol. 30, pp. 79–85, 2016, ISSN: 1746-8094. DOI: https://doi.org/10.1016/j.bspc.2016.06.017.

[19] W. De Clercq, A. Vergult, B. Vanrumste, W. Van Paesschen, and S. Van Huffel, "Canonical Correlation Analysis Applied to Remove Muscle Artifacts from the Electroencephalogram," en, *IEEE Trans Biomed Eng*, vol. 53, no. 12 Pt 1, pp. 2583–2587, Dec. 2006.

[20] P. Gajbhiye, R. K. Tripathy, A. Bhattacharyya, and R. B. Pachori, "Novel Approaches for the Removal of Motion Artifact From EEG Recordings," *IEEE Sensors Journal*, vol. 19, no. 22, pp. 10 600–10 608, 2019. DOI: 10.1109/JSEN.2019.2931727.

[21] A. K. Waljee and P. D. Higgins, "Machine Learning in Medicine: A Primer for Physicians," *Official journal of the American College of Gastroenterology—ACG*, vol. 105, no. 6, pp. 1224–1226, 2010.

[22] S. Mahmud, M. S. Hossain, M. E. H. Chowdhury, and M. B. I. Reaz, "MLMRS-Net: Electroencephalography (EEG) Motion Artifacts Removal Using a Multi-Layer Multi-Resolution Spatially Pooled 1D Signal Reconstruction Network," *Neural Computing and Applications*, vol. 35, no. 11, pp. 8371–8388, Apr. 2023, ISSN: 1433-3058. DOI: 10.1007/s00521-022-08111-6.

[23] N. A. Alzahab, L. Apollonio, A. D. Iorio, *et al.*, "Hybrid Deep Learning (hDL)-Based Brain-Computer Interface (BCI) Systems: A Systematic Review," *Brain Sciences*, vol. 11, no. 1, Jan. 2021. DOI: 10.3390/brainsci11010075.

[24] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds., Cham: Springer International Publishing, 2015, pp. 234–241, ISBN: 978-3-319-24574-4.

[25] S. Wang, O. Wang, R. Zhang, A. Owens, and A. A. Efros, "CNN-Generated Images are Surprisingly Easy to Spot... for Now," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. abs/1912.11035, Jun. 2020.

[26] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, "1D Convolutional Neural Networks and Applications: A Survey," *Mechanical Systems and Signal Processing*, vol. 151, p. 107 398, 2021, ISSN: 0888-3270. DOI: https://doi.org/10.1016/j.ymssp.2020.107398.

[27] J. Malik, S. Kiranyaz, and M. Gabbouj, "Self-Organized Operational Neural Networks for Severe Image Restoration Problems," *Neural Networks*, vol. 135, pp. 201–211, 2021, ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2020.12.014.

[28] M. N. M. Sandhyalati Behera, "A Machine Learning Approach for Artifact Removal from Brain Signal," *Computer Systems Science and Engineering*, vol. 45, no. 2, pp. 1455–1467, 2023. DOI: 10.32604/csse.2023.029649.

[29] M. Klados and P. Bamidis, "A Semi-Simulated EEG/EOG Dataset for the Comparison of EOG Artifact Rejection Techniques," *Data in Brief*, vol. 8, Jun. 2016. DOI: 10.1016/j.dib.2016.06.032.

[30] X. Navarro, F. Porée, and G. Carrault, "ECG Removal in Preterm EEG Combining Empirical Mode Decomposition and Adaptive Filtering," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 661–664. DOI: 10.1109/ICASSP.2012.6287970.

[31] S. S. Priyadharsini and S. E. Rajan, "An Efficient Method for the Removal of ECG Artifact from Measured EEG Signal using PSO Algorithm," *International Journal of Advances in Soft Computing and its Applications*, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:1900869.

[32] C. Dai, J. Wang, J. Xie, W. Li, Y. Gong, and Y. Li, "Removal of ECG Artifacts From EEG Using an Effective Recursive Least Square Notch Filter," *IEEE Access*, vol. 7, pp. 158 872–158 880, 2019. DOI: 10.1109/ACCESS.2019.2949842.

[33] S. A. M. Mane and A. Shinde, "StressNet: Hybrid Model of LSTM and CNN for Stress Detection from Electroencephalogram Signal (EEG)," *Results in Control and Optimization*, vol. 11, p. 100 231, 2023, ISSN: 2666-7207. DOI: https://doi.org/10.1016/j.rico.2023.100231.

[34] J. Wang, S. Cheng, J. Tian, and Y. Gao, "A 2D CNN-LSTM Hybrid Algorithm Using Time Series Segments of EEG Data for Motor Imagery Classification," *Biomedical Signal Processing and Control*, vol. 83, p. 104 627, 2023, ISSN: 1746-8094. DOI: https://doi.org/10.1016/j.bspc.2023.104627.

[35] R. Naily, S. Yahia, and M. Zaied, "A New Deep Learning Architecture Based on LSTM and Wavelet Transform for Epileptic EEG Signal Classification," in *Intelligent Systems Design and Applications*, A. Abraham, A. Bajaj, T. Hanne, and P. Siarry, Eds., Cham: Springer Nature Switzerland, 2024, pp. 353–362, ISBN: 978-3-031-64813-7.

[36] Y. Pang, J. Lin, T. Qin, and Z. Chen, "Image-to-Image Translation: Methods and Applications," *IEEE Transactions on Multimedia*, vol. 24, pp. 3859–3881, 2022. DOI: 10.1109/TMM.2021.3109419.

[37] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-To-Image Translation With Conditional Adversarial Networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017.

[38] A. Alotaibi, "Deep Generative Adversarial Networks for Image-to-Image Translation: A Review," *Symmetry*, vol. 12, no. 10, 2020, ISSN: 2073-8994. DOI: 10.3390/sym12101705.

[39] D. Torbunov, Y. Huang, H. Yu, *et al.*, *UVCGAN: UNet Vision Transformer cycle-consistent GAN for unpaired image-to-image translation*, Jan. 2023.

[40] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://iclr.cc/virtual/2021/oral/3458.

[41] J. Chen, J. Mei, X. Li, *et al.*, "TransUNet: Rethinking the U-Net Architecture Design for Medical Image Segmentation Through the Lens of Transformers," *Medical Image Analysis*, vol. 97, p. 103 280, 2024, ISSN: 1361-8415. DOI: https://doi.org/10.1016/j.media.2024.103280.

[42] A. Vaswani, "Attention is All You Need," *Advances in Neural Information Processing Systems*, 2017.

[43] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, *et al.*, "Generative Adversarial Networks," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., vol. 27, Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2014/file/f033ed80deb0234979a61f95710dbe25-Paper.pdf.

[44] I. Goodfellow, J. Pouget-Abadie, M. Mirza, *et al.*, "Generative Adversarial Networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, Oct. 2020, ISSN: 0001-0782. DOI: 10.1145/3422622.

[45] P. Salehi, A. Chalechale, and M. Taghizadeh, "Generative Adversarial Networks (GANs): An Overview of Theoretical Model, Evaluation Metrics, and Recent Developments," *arXiv Preprint*, vol. abs/2005.13178, 2020. arXiv: 2005.13178. [Online]. Available: https://arxiv.org/abs/2005.13178.

[46] A. Jabbar, X. Li, and B. Omar, "A Survey on Generative Adversarial Networks: Variants, Applications, and Training," *ACM Computing Surveys*, vol. 54, no. 8, Oct. 2021, ISSN: 0360-0300. DOI: 10.1145/3463475.

[47] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2242–2251. DOI: 10.1109/ICCV.2017.244.

[48] T. Miyato and M. Koyama, "cGANs with Projection Discriminator," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=ByS1VpgRZ.

[49] E. Schonfeld, B. Schiele, and A. Khoreva, "A U-Net Based Discriminator for Generative Adversarial Networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020.

[50] A. Khorshidifar, G. Mostaghel, K. Dastvareh, Y. Ahmadyar, and R. Samimi, "A CycleGAN-Based Method for Generating Virtual Subtracted Images in Dual-Energy Contrast-Enhanced Spectral Mammography," *Journal of Nuclear Medicine*, vol. 65, no. supplement 2, pp. 241 398–241 398, 2024, ISSN: 0161-5505. [Online]. Available: https://jnm.snmjournals.org/content/65/supplement_2/241398.

[51] H. Thanh-Tung, T. Tran, and S. Venkatesh, "Improving Generalization and Stability of Generative Adversarial Networks," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=ByxPYjC5KQ.

[52] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved Training of Wasserstein GANs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Long

Beach, California, USA: Curran Associates Inc., 2017, pp. 5769–5779, ISBN: 9781510860964.

[53] S.-H. Tsang, *Review — SimGAN: Learning from Simulated and Unsupervised Images through Adversarial Training (GAN) — by Sik-Ho Tsang — Medium*, 2021. [Online]. Available: https://sh-tsang.medium.com/review-simgan-learning-from-simulated-and-unsupervised-images-through-adversarial-training-gan-86a7003add50.

[54] D. Torbunov, Y. Huang, H.-H. Tseng, *et al.*, "UVCGAN v2: An Improved Cycle-Consistent GAN for Unpaired Image-to-Image Translation," *arXiv Preprint*, Mar. 2023. [Online]. Available: https://arxiv.org/abs/2303.16280v3.

[55] P. S. Vadar, T. T. Moharekar, and U. R. Pol, "A Comprehensive Comparison of Deep Learning Libraries: TensorFlow, PyTorch, FastAI, Keras, and PyTorch Lightning," *International Journal of Computer Science Engineering Techniques*, vol. 8, 6 2024.

[56] SciPy, *Detrend — SciPy v1.14.1 Manual*. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.detrend.html.

[57] SciPy, *Pearsonr — SciPy v1.14.1 Manual*. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.pearsonr.html.

[58] BrainFlow, *BrainFlow Documentation*. [Online]. Available: https://brainflow.readthedocs.io/en/stable/index.html.

[59] Numpy, *Numpy.polynomial.polynomial.Polynomial.fit — NumPy v2.1 Manual*. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.polynomial.polynomial.Polynomial.fit.html.

[60] SciPy, *Iirnotch — SciPy v1.14.1 Manual*. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.iirnotch.html.

[61] Z. W. Tan, *Deep Learning Based Speech Enhancement for Noise Adverse Environment*, 2024. DOI: 10.32657/10356/178275.

[62] P. Zhou, B. Schwerin, and S. So, "U-Net Based Fetal R-peak Prediction From Abdominal ECG Signals," in *2024 9th International Conference on Signal and Image Processing (ICSIP)*, 2024, pp. 121–125. DOI: 10.1109/ICSIP61881.2024.10671460.

[63] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML'10, Haifa, Israel: Omnipress, 2010, pp. 807–814, ISBN: 9781605589077.

[64] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[65] S. Bhakta, U. Nandi, K. Ray Mahapatra, M. Marjit Singh, and A. Noorwali, "SWOSBC: A Novel Optimizer for Learning Convolutional Neural Network," *IEEE Access*, vol. 12, pp. 156 458–156 470, 2024. DOI: 10.1109/ACCESS.2024.3481640.

[66] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature Pyramid Networks for Object Detection," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 936–944. DOI: 10.1109/CVPR.2017.106.

[67] G. Ghiasi, T.-Y. Lin, and Q. V. Le, "NAS-FPN: Learning Scalable Feature Pyramid Architecture for Object Detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 7036–7045.

[68] Y. Gong, X. Yu, Y. Ding, X. Peng, J. Zhao, and Z. Han, "Effective Fusion Factor in FPN for Tiny Object Detection," in *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, 2021, pp. 1160–1168.

[69] X. Zhou and L. Zhang, "SA-FPN: An Effective Feature Pyramid Network for Crowded Human Detection," *Applied Intelligence*, vol. 52, no. 11, pp. 12 556–12 568, 2022.

[70] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[71] PyTorch, *TransformerEncoder — PyTorch 2.5 documentation*. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.TransformerEncoder.html.

[72] S. Kiranyaz, T. Ince, A. Iosifidis, and M. Gabbouj, "Operational Neural Networks," *Neural Computing and Applications*, vol. 32, no. 11, pp. 6645–6668,

Jun. 2020, ISSN: 1433-3058. DOI: 10.1007/s00521-020-04780-3. [Online]. Available: https://doi.org/10.1007/s00521-020-04780-3.

[73] S. Kiranyaz, J. Malik, H. B. Abdallah, T. Ince, A. Iosifidis, and M. Gabbouj, "Self-Organized Operational Neural Networks with Generative Neurons," *Neural Networks*, vol. 140, pp. 294–308, 2021, ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2021.02.028.

[74] J. Malik, S. Kiranyaz, and M. Gabbouj, *FastONN – Python Based Open-Source GPU implementation for Operational Neural Networks*, 2020. arXiv: 2006.02267 [cs.NE]. [Online]. Available: https://arxiv.org/abs/2006.02267.

[75] D. I. Kosasih, B.-G. Lee, and H. Lim, "Multichannel One-Dimensional Data Augmentation with Generative Adversarial Network," *Sensors*, vol. 23, no. 18, 2023, ISSN: 1424-8220. DOI: 10.3390/s23187693.

[76] O. Mujahid, I. Contreras, A. Beneyto, I. Conget, M. Giménez, and J. Vehi, "Conditional Synthesis of Blood Glucose Profiles for T1D Patients Using Deep Generative Models," *Mathematics*, vol. 10, no. 20, 2022, ISSN: 2227-7390. DOI: 10.3390/math10203741.

[77] S. Zia and A. Nawaz Khan, *Activities of Daily livings using a Portable EEG Headset*, 2021. DOI: 10.21227/mtwx-p951.

[78] M. Torkamani-Azar, S. D. Kanik, S. Aydin, and M. Cetin, "Prediction of Reaction Time and Vigilance Variability From Spatio-Spectral Features of Resting-State EEG in a Long Sustained Attention Task," *IEEE Journal of Biomedical and Health Informatics*, vol. 24, no. 9, pp. 2550–2558, 2020. DOI: 10.1109/JBHI.2020.2980056.

[79] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=Bkg6RiCqY7.

[80] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[81] G. Schalk, D. J. McFarland, T. Hinterberger, N. Birbaumer, and J. R. Wolpaw, "BCI2000: A General-Purpose Brain-Computer Interface (BCI) System," *IEEE Transactions on Biomedical Engineering*, vol. 51, no. 6, pp. 1034–1043, Jun. 2004.