UHASSELT
KNOWLEDGE IN ACTION

Maastricht University

Faculteit Wetenschappen
School voor Informatietechnologie

master in de informatica

Masterthesis

From Individual to Generalized: Multi-Subject Training in State-of-the-Art Finger Movement Reconstruction BCI Models

Ward Ceyssens

Scriptie ingediend tot het behalen van de graad van master in de informatica

PROMOTOR :

Prof. dr. Stijn VANSUMMEREN

BEGELEIDER :

dr. ir. Axel FAES

2024
2025

De transnationale Universiteit Limburg is een uniek samenwerkingsverband van twee universiteiten in twee landen: de Universiteit Hasselt en Maastricht University.

UHASSELT
KNOWLEDGE IN ACTION

www.uhasselt.be
Universiteit Hasselt
Campus Hasselt:
Martelarenlaan 42 | 3500 Hasselt
Campus Diepenbeek:
Agoralaan Gebouw D | 3590 Diepenbeek

Maastricht University

Faculteit Wetenschappen
School voor Informatietechnologie

master in de informatica

Masterthesis

From Individual to Generalized: Multi-Subject Training in State-of-the-Art Finger Movement Reconstruction BCI Models

Ward Ceyssens

Scriptie ingediend tot het behalen van de graad van master in de informatica

PROMOTOR :

Prof. dr. Stijn VANSUMMEREN

BEGELEIDER :

dr. ir. Axel FAES

Acknowledgments

I would like to thank my promotor, Professor Stijn Vansummeren, and my mentor, Axel Faes, for their invaluable assistance throughout this project. The authors of the Fingerflex and DeepInsight papers for publishing the code for their models online, which helped immensely in reproducing the papers. I also need to thank Axel Faes again for taking the time to write an implementation of his BTTR paper for me to use. And finally Xie Ziqian for responding to my emails and answering my questions regarding the paper *Decoding of finger trajectory from ECoG using deep learning* (Xie, Schwartz, and Prasad 2018).

Additionally I would like to thank my family for their support, and my friends for putting up with my lack of free time these last few months.

Summary

Brain Computer Interfaces(BCI) attempt to map activity in the human brain to digital signals for various purposes. In this work we specifically focus on BCI methods which attempt to translate brain signals into finger movements.

A major obstacle for these methods is that the procedure to implant the electrodes needed to record this brain activity is quite invasive, which severely limits the amount of available datasets. The datasets currently available contain only 10 minutes of recorded data per subject.

In this work we evaluate the possibility of using the data of other test subjects to expand the amount of data these models can use to learn from, and possibly allow for the creation of a general model that is not specific to an individual test subject in its predictions. We evaluate two state-of-the-art methods, an older model which was likely to work well, and one we created ourselves.

The FingerFlex model is a deep learning autoencoder, and had the most promising results: model performance increased slightly and a general model only needed a fraction of the normal training time to function on a new subject.

The second model uses Block-Term Tensor Decomposition and had poor results, performance was lowered and the general model is only slightly better than random guesses.

The third model is a convolutional model meant to be flexible to be used for many different BCI applications, and was broadly similar to FingerFlex. This model performed extremely poorly when finetuned.

Finally our own model based on gradient boosting had the opposite results of FingerFlex: generally lowered performance, and retraining the general model on a subject took longer than simply training one from scratch.

We believe the main obstacle to be the lack of standardization of the brain signal recording between subjects, as the number and placement of electrodes varies wildly.

Samenvatting

Brain Computer Interfaces (BCI) proberen activiteit in de menselijke hersenen om te zetten in digitale signalen voor verschillende doeleinden. In dit werk richten we ons specifiek op BCI-methoden die hersensignalen proberen te vertalen naar vingerbewegingen.

Een groot obstakel voor deze methoden is dat de procedure voor het implanteren van de elektroden die nodig zijn om deze hersenactiviteit te registreren, vrij invasief is, wat de hoeveelheid beschikbare datasets ernstig beperkt. De momenteel beschikbare datasets bevatten slechts 10 minuten aan opgenomen data per proefpersoon.

In dit werk evalueren we de mogelijkheid om de data van andere proefpersonen te gebruiken om de hoeveelheid data die deze modellen kunnen gebruiken om van te leren uit te breiden, en mogelijk de creatie van een algemeen model mogelijk te maken dat niet specifiek is voor een individuele proefpersoon in zijn voorspellingen. We evalueren twee state-of-the-art methoden: een ouder model dat waarschijnlijk goed zou werken, en een model dat we zelf hebben ontwikkeld.

Het FingerFlex-model is een deep learning auto-encoder en had de meest veelbelovende resultaten: de modelprestaties verbeterden licht en een algemeen model had slechts een fractie van de normale trainingstijd nodig om te functioneren op een nieuwe proefpersoon.

Het tweede model maakt gebruik van Block-Term Tensor Decomposition en leverde slechte resultaten op. De prestaties waren lager en het algemene model was slechts iets beter dan willekeurige schattingen.

Het derde model is een convolutioneel model dat flexibel is en gebruikt kan worden voor veel verschillende BCI-toepassingen, en was grotendeels vergelijkbaar met FingerFlex. Dit model presteerde extreem slecht na finetuning.

Ten slotte had ons eigen model, gebaseerd op gradient boosting, de tegenovergestelde resultaten van FingerFlex: de prestaties waren over het algemeen lager en het opnieuw trainen van het algemene model op een proefpersoon duurde langer dan het simpelweg vanaf nul trainen van een model.

Wij denken dat het grootste obstakel het gebrek aan standaardisatie van de hersensignaalregistratie tussen proefpersonen is, aangezien het aantal en de plaatsing van elektroden sterk uiteenlopen.

Contents

Chapter 1

Introduction

Brain-computer interfaces(BCI) are systems which use the activity of the human brain to direct machines without needing the person to physically move a muscle. Essentially using your mind to control machines. BCI has a lot of overlap with the field of neuroprosthetics, which is concerned with using artificial devices to restore the function of sensory organs, nervous systems, or other issues relating to the brain.

The most common neuroprosthetic is the cochlear implant, which helps people with hearing loss regain function. The implant consists of an external microphone and processing component and an internal implant which stimulates the cochlear nerve directly to send an auditory signal to the brain. Through training and regular use, patients with a cochlear implant can learn to interpret those signals as recognizable sounds and speech.

Visual prosthetics have had success by stimulating the retina, optical nerve, or visual cortex to provide a primitive form of vision, though current methods have a low resolution.

Communication is another common application of BCI, with spelling systems based on the brain's reaction to seeing the special character highlighted (Pan et al. 2022), systems that turn imagined handwriting to text (Willett et al. 2021), and even imagined speech-to-text (Metzger et al. 2023).

Finally, there has been success in allowing tetrapalegic patients to control a robotic arm and perform other tasks (Hochberg et al. 2006).

Beyond restoring lost function, BCI can also be used to control machines which do not replicate existing body parts. For example the tetrapalegic patient from Hochberg et al. 2006 was also able to control a cursor on a computer screen, allowing him to open emails and control a television, even while having a conversation.

Generally, the process through which a BCI works involves four steps.

First, measurements are taken of brain activity. Some techniques include EEG (Electroencephalogram), fMRI (Functional Magnetic Resonance Imaging), MEG (Magneto Encephalography), NIRS (Near-infrared Spectroscopy), PET (Positron Emission Tomography), and EROS (Event related optical signal) (Vaid, Singh, and Kaur 2015)

Once these signals are collected, they must be processed to remove noise and recording artifacts. Other signal processing techniques such as converting the signals to time-frequency representations may also be used depending on what works best for the next step.

After signal processing, feature extraction is performed. The aim here is to reduce the data into a smaller amount of extracted features upon which classification can be performed.

Finally, a command is then given to the external device, based on the classification of the signal.

An optional fifth step is feedback, where the user either sees something happen or the BCI component sends a signal to the brain. This feedback step allows the user to adapt to the inaccuracies of the BCI component and correct errors. Obviously feedback is not possible when working on prerecorded data instead of a live prototype.

From 2001 to 2008 a series of BCI competitions were held by the Berlin Brain-Computer Interface (BBCI) to encourage research and provide datasets for BCI applications. One of the challenges in the fourth competition was to predict finger movements from ECoG data (Schalk et al. 2007). ECoG involves electrodes placed directly on the brain and thereby provides excellent spatial accuracy, a more detailed explanation can be found in section 2.1. The finger movements to predict consist of just the flex of the finger as measured by a dataglove. This challenge mainly serves to help develop BCI solutions to distinguish more detailed movements.

The published challenge received five submissions during the initial competition and in the years since many papers have come out improving on the performance of previous methods. While the current state-of-the-art models have reasonably high accuracy, many still require extensive training time or use methods that would be unwieldy to use in real-time. In addition, due to the relative invasiveness of implanting ECoG electrodes there is a lack of easily available datasets, the BCI competition dataset only providing a 10 minute long recording for each of its three subjects. When this ten minute dataset then has to be even further split into training, testing, and validation sections, this lack of data clearly becomes a major obstacle. With an eye towards practical uses it would also be ideal to reduce the "calibration" period a user would need to wait while the model gathers data and is trained.

The purpose of this thesis is therefore to evaluate how well current methods perform when trained on data of multiple subjects, and then applied to an unseen subject. Training on multiple subjects greatly increases the amount of available training data for the model to learn from, and if a model could be generalized to work on any patient with less or no finetuning needed it would greatly improve and expand upon the practical uses of the model.

In chapter 2, we will explain some background knowledge needed to understand how the data this BCI application works with has been collected, how it's structured, and the metric used to judge the performance of these models. Chapter 3 provides an overview of the preexisting literature on the subject, and summarizes most of the proposed models applicable to the BCI IV challenge. In chapter 4 we explain our methodology for evaluating these models and provide a more detailed accounting of the state-of-the-art models we chose to evaluate. It also elaborates on any modifications in our reproductions which differ from the original. Chapter 5 discusses the results of our experiments. In chapter 6 we discuss some initial experiments we performed to try and standardize the ECoG data between subjects in order to improve model performance. In chapter 7 we conclude that some models show promising reductions in the training time needed, but that other models lose performance. Chapter 8 presents several promising directions of further study, and a few prospective methods to standardize subjects better in order to improve the performance of the general model.

Chapter 2

Background

2.1 Electrocorticography (ECoG)

The brain signals in the dataset this work uses were recorded using electrocorticography (or ECoG for short). ECoG is a method of measuring electrical activity in the brain by placing electrodes directly on the surface of the brain. This gives it a great advantage over conventional electroencephalography (EEG) as the electrical signals no longer need to pass through the skull where the low conductivity of bone and rapid attenuation produces a "smearing effect" (Hashiguchi et al. 2007). As a result ECoG has a higher spatial resolution than EEG at the cost of requiring a more invasive procedure to implant the electrode grids.

ECoG is commonly used in the treatment of epilepsy to perform detailed mapping of epileptogenic regions and lesions before surgery and to monitor the success of the surgery by detecting any remaining epileptiform activity (Sugano, Shimizu, and Sunaga 2007). Aside from clinical applications, the high resolution of ECoG allows for real-time functional brain mapping which has made it a promising technique for the development of brain-computer interfaces (BCI) such as those discussed in this text.

Since implanting ECoG electrodes is a rather invasive procedure, it is difficult to justify implanting them just for BCI studies. Many recordings come from patients who already had ECoG grids implanted for medical purposes such as brain mapping and as a result the electrode locations and amounts vary wildly between patients. Figure 2.1 shows an example of an ECoG electrode grid implanted on a subject.

2.2 Pearson correlation

The performance of the techniques discussed in this paper is measured by the Pearson correlation coefficient - often shortened to simply *the correlation* - between the predicted finger flexions and the actual finger flexions. Pearson correlation is calculated as

$$\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y}$$

where

- X and Y are random variables
- *cov* is the covariance
- σ_X is the standard deviation of X.
- σ_Y is the standard deviation of Y.

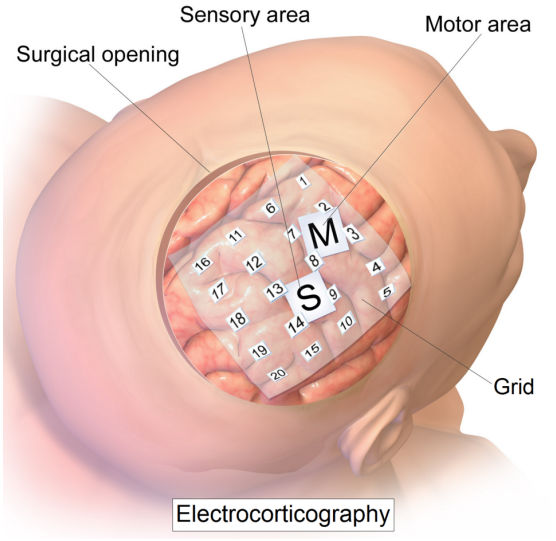


Figure 2.1: ECoG electrode grid

Source: Blausen.com staff 2014

The Pearson correlation coefficient always has a value between -1 and 1, with 0 implying the two variables have no linear dependency. A value of 1 would imply complete correlation, if X increases Y increases by the same relative value, and would be the optimal result. A negative value implies our model predicts the opposite of what actually occurs, if X increases Y decreases, and this is easily turned into a positive correlation by flipping the output of our model.

Pearson correlation has the advantage of measuring the linear relation between two variables while being unaffected by scaling or offsets.

2.3 Dataset

In order to compare how the chosen techniques perform when pretrained on subjects we need to have a dataset with multiple different subjects. Most papers on this subject use the BCI Competition IV (Schalk et al. 2007) dataset which consists of only 3 different test subjects. This would leave only two other subjects to train on which might be insufficient. Luckily Miller (2019) provides data collected in a similar manner for 9 different subjects.

Each subject had subdural electrode grids implanted providing 38 to 64 channels of ECoG data depending on the amount of electrodes in the grid. Subjects were fitted with a dataglove sensor to measure finger flexion. As such the dataset consists of ECoG electrode measurements and dataglove finger flexion measurements. The ECoG data is the input used by the models discussed in this work to predict the finger flexion output.

Patients were cued to move an indicated finger for a period of 2 seconds followed by a period of rest, with a total of 30 cues for each finger interleaved randomly. Rest periods of 2 seconds followed each movement trial.
Each subject has been given a 2 letter patient code and this code will be used throughout this text to refer to a specific subject from the dataset.

The ECoG data was sampled at 1000 Hz and scaled such that one unit corresponds to .0298 microvolts. Finally a 1-pole band-pass filter was applied from 0.15 to 200 Hz, to filter out everything outside that range.
The finger flexion data contains five channels corresponding to the thumb-,index-,middle-,ring-, and little finger in order. Each channel is in 40 ms blocks with non-zero offset.

Patient Code	Subject	Age	Sex	Handedness	Array location	# Electrodes
bp	1	18	F	R	L Frontoparietal	46
cc	2	21	M	R	R Frontotemporal	63
zt	3	27	F	R	L Fronto-temporal-parietal	61
jp	4	35	F	R	L Fronto-Temporal	58
ht	5	26	M	R	L Parietal - Temporal - Occipital	64
mv	6	45	F	R	L Frontotemporal	43
wc	7	32	M	R	L Fronto-temporal-parietal	64
wm	8	19	F	R	R Fronto-parietal	38
jc	9	18	F	R	L Frontal	47

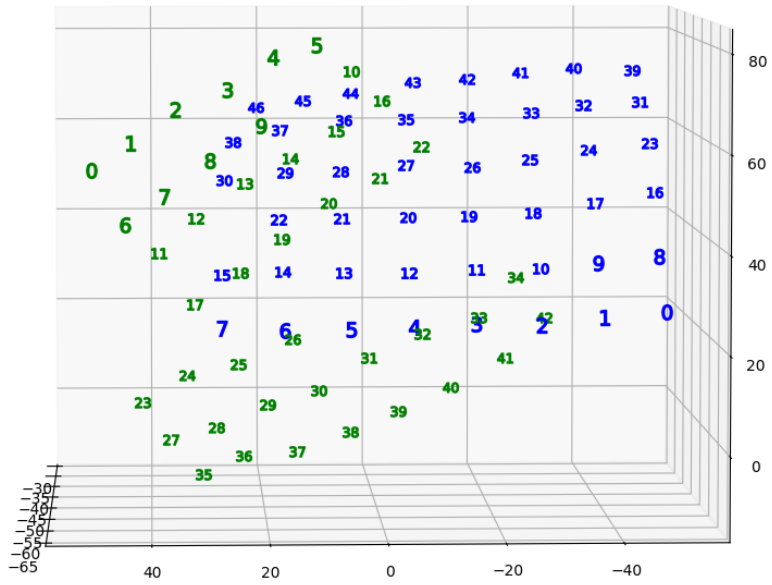
Table 2.1: Fingerflex subjects

The dataset also includes additional useful information such as the location of the electrodes and the timing of the movement cues shown to the patients, but since this information was intentionally left out of the BCI Competition dataset we will not make use of this information. Figure 2.2 plots some of the electrodes in 3d space, showing off the largest obstacle to training a general model. While all subjects in the dataset have the grid covering the section of the brain responsible for finger movements, the ordering and location of electrodes varies wildly. Some, like fig. 2.2b mostly match if the order of electrodes is reversed but others like fig. 2.2a have a different form factor, meaning the 8th electrode is at the end of a row in one subject but in the middle of a row in another.

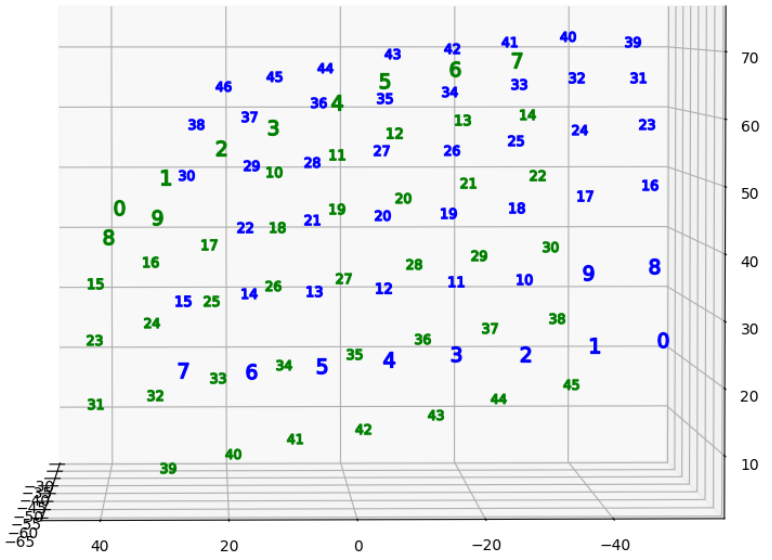
Figure 2.3 shows a section of the data of subject bp, with fig. 2.3a showing the first five electrode data channels and fig. 2.3b displaying the corresponding recorded finger flexion data. Table 2.1 lists details about each subject.

Subjects cc and wm have been excluded from our testing set due to fears that including data from an entirely different section of the brain could confuse the model and wm's low electrode count.

> **Ethics statement:** All patients participated in a purely voluntary manner, after providing informed written consent, under experimental protocols approved by the Institutional Review Board of the University of Washington (#12193). All patient data was anonymized according to IRB protocol, in accordance with HIPAA mandate. These data originally appeared in the manuscript "Human Motor Cortical Activity Is Selectively Phase- Entrained on Underlying Rhythms" published in PLoS Computational Biology in 2012 (Miller et al. 2012).

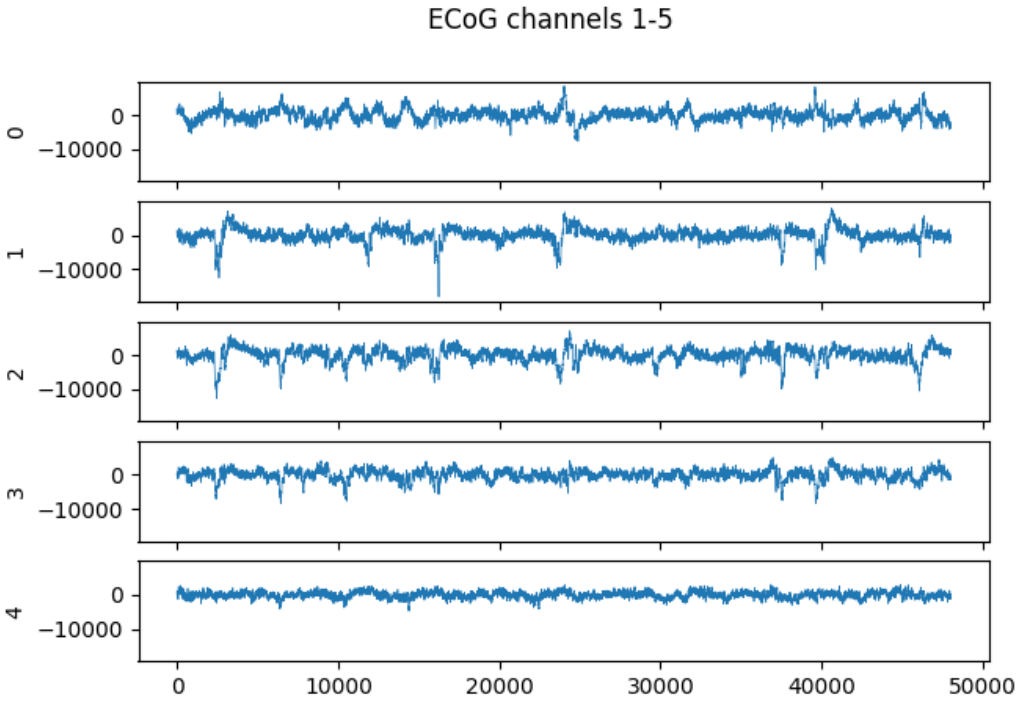


(a) Electrode positions of subjects jc(blue) and mv(green)

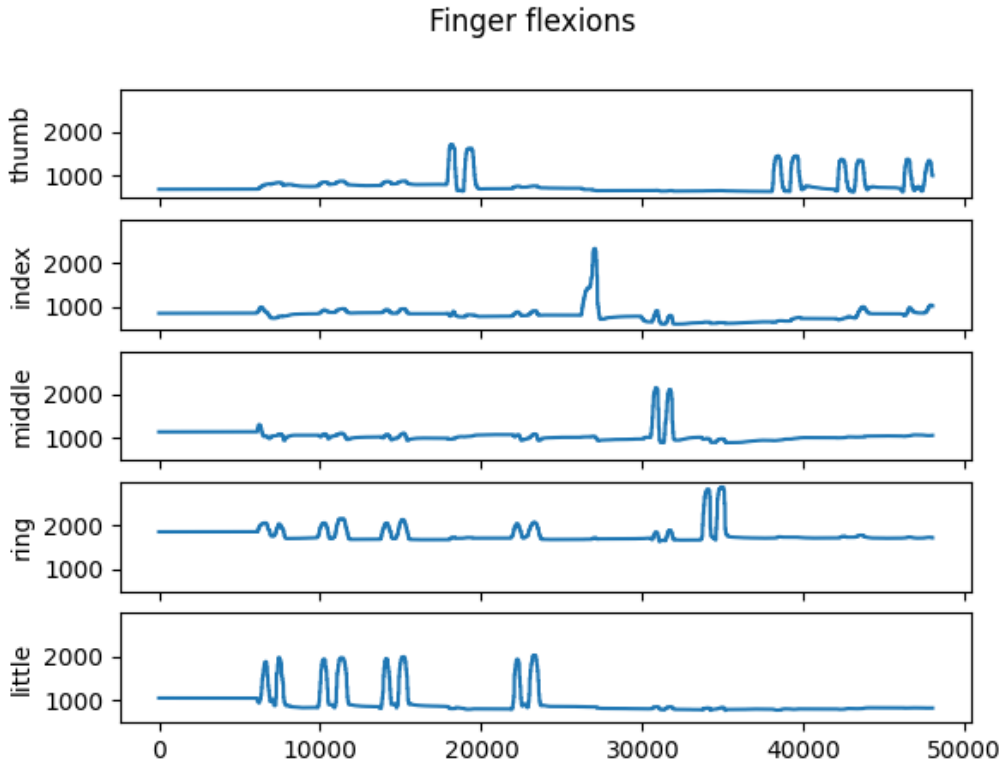


(b) Electrode positions of subjects jc(blue) and bp(green)

Figure 2.2: Electrode position plots, numbered by channel order



(a)



(b)

Figure 2.3: Sample data from subject bp

Chapter 3

Literature

In this section we will summarize the models which were submitted in the initial competition, some of the improved models which have been proposed since then, and finally the currently best performing models.

3.1 BCI competition IV

The first usage of this dataset is when it was published as a challenge for the fourth BCI competition held by the Berlin Brain-Computer Interface (BBCI) research group. The main goal of this challenge was to use the high spatial resolution ECoG provides to distinguish very closely clustered sections of the brain, such as the motor cortex sections controlling individual finger movements.
Two of the submitted solutions were published as *Decoding finger movements from ECoG signals using switching linear models* (Flamary and Rakotomamonjy 2012) and *Decoding finger flexion from band-specific ECoG signals in humans* (Liang and Bougrain 2012). Respectively they achieved a correlation of 0.42 and 0.46. Or 0.43 and 0.48 if the correlation of all fingers is included in the average. The original BCI competition excluded the fourth (ring) finger from the total because it's flexion was highly correlated to that of the third (middle) and fifth (little) fingers. We will use the average correlation of all fingers from this point on, as do most research papers.

Decoding finger movements from ECoG signals using switching linear models makes use of the fact that the dataset has the subject move only one finger at a time, and switches between six linear models depending on which finger the switching function predicts to be the correct one. The sixth model represents no finger movement. It should be noted that each model still predicts the flex of all five fingers, since movement of one finger often results in correlated movement of other fingers.
For preprocessing they make use of an auto-regressive model, downsampling, and an algorithm to select only the most relevant channels from the full selection. In addition to their 0.43 correlation, they report a correlation of 0.61 when the model is provided with the correct finger. While that is only possible by already knowing the finger flexions, it shows that estimating the correct finger is the main limiting factor for this model.

Decoding finger flexion from band-specific ECoG signals in humans is heavily inspired by the band decomposition and amplitude modulation proposed in Sanchez et al. 2008.
They decompose each electrode channel into three frequency bands: the sub-bands (1–60 Hz), the gamma band (60–100 Hz), and the fast gamma band (100–200 Hz) after which amplitude modulation(AM) is applied over a time window of 40 ms such that the AM features have the same sampling rate as the finger flexion data. Since this would create a large amount of feature channels they employed a forward feature selection procedure which, starting from the empty

set of features, added the feature which most increased correlation on the validation set until either the correlation no longer increases or a given number of features is reached. This number was set to 10 features but correlations ceased to increase for the test set after the first four features were selected.
The model itself is a simple linear regression model.

3.2 Further Research

These are some of the models that have been proposed after the fourth BCI competition ended but whose performance has since been eclipsed by newer models.

Decoding of finger trajectory from ECoG using deep learning (Xie, Schwartz, and Prasad 2018) improves on the state-of-the-art by using deep learning instead of the previous linear models.
For preprocessing some channels are removed due to high noise. The finger flexion trajectories are cleaned up by correcting the baseline, thresholding to set finger flexion in rest states to zero, and setting all non-maximum finger flexions to zero to ensure only one finger is moving during the movement sections. The input is provided in windows of four seconds, or 100 time steps.
A convolutional neural network is used as a hierarchical feature extractor which feeds into a recurrent neural network which can process temporal features. The first layer of the model is 1x1 spatial convolution, followed by three layers of temporal convolution. An L_2 pooling layer follows and rescales the features. Next is a Long short term memory (LSTM) layer and finally a dense layer provides a prediction at every timestep of the inputs window.
This model achieves an average correlation of 0.52. This model was selected as the fourth model to be evaluated in this work but was not included due to being unable to replicate it in the limited time available.

Interpreting wide-band neural activity using convolutional neural networks (Frey et al. 2021) proposes a much more generalized convolutional neural network for use in a variety of BCI experiments, which they apply to calcium imaging in rats along with the ECoG recordings we focus on.
Preprocessing involves morlet wavelet transformation, followed by channel and frequency wise normalization using median absolute deviation (MAD).
The model consists of 17 layers, the first 1–8 layers share the weights over the channel dimension while layers 9–15 share the weights across the time dimension. These convolution layers serve to downsample the number of channels and some may be added or removed when the input has a different number of channels. The last two layers are fully connected layers.
They report an average correlation of 0.52, but this number excludes the fourth finger and the paper does not provide the per-finger correlation needed to calculate the average of all fingers. This paper was selected as the third method to be evaluated, due to the similarity to Fingerflex and we considered that the design intention for the model to be flexible with minimal preprocessing or assumptions about the input data may translate well to the metrics being evaluated in this work.

Fast and accurate decoding of finger movements from ECoG through Riemannian features and modern machine learning techniques (Yao, Zhu, and Shoaran 2022) uses gradient-boosted trees and Riemannian-space features to predict finger flexions.
The ECoG data was common average referenced and divided into epochs of 200 ms. From which they, in addition to Riemannian features, extract alpha (8–13 Hz), beta (13–30 Hz), low-gamma (30–60 Hz), gamma (60–100 Hz), and high-gamma (100–200 Hz) power, local motor potentials, the Hjorth activity, mobility, and complexity parameters. Features from multiple epochs are concatenated to provide additional temporal information. This resulted in a large number of features so feature selection was used based on the squared Pearson correlation coefficient.
The model achieved an average correlation of 0.54. It was considered for testing in this work but was not included due to complexity and time constraints.

3.3 State-of-the-art

This section summarizes the state-of-the-art models we have chosen to evaluate. At the time of writing they are some of the highest performing models.

Single Finger Trajectory Prediction From Intracranial Brain Activity Using Block-Term Tensor Regression With Fast and Automatic Component Extraction (Faes, Camarrone, and Van Hulle 2024) presents a multiway tensor decomposition regression algorithm. While it's main advantage is the vastly decreased computational requirements it still performs well with an average Pearson correlation of 0.59 for the three subjects from the BCI competition dataset.
This algorithm was selected for this thesis due to it's difference from more conventional machine learning methods, to ensure a variety of methods are evaluated. Additionally, Dr. Ir. Axel Faes is mentoring this thesis and was willing to provide us with an implementation of BTTR.
For preprocessing they normalize and Common Average Reference (Ludwig et al. 2009) both the ECoG and finger flexion data. They split each ECoG channel into eight sub-bands: δ(1.5 -5 Hz), θ(5 -8 Hz), α(8 -12Hz), β_1(12 -24 Hz), β_2(24 -34 Hz), γ_1(34 -60 Hz), γ_2(60 - 100 Hz), and γ_3(100 - 130 Hz). They then filters out resting segments from the data by removing data points more than one second before movement. Finally they downsample the ECoG data to 10 Hz.
The model itself uses Block Tensor decomposition along with a proposed algorithm to automatically select the correct rank for each decomposition component called Automatic Component Extraction (ACE), and its improved version augmented with automatic latent component selection, called Automatic Correlated Component Selection (ACCoS).

Fingerflex: Inferring Finger Trajectories from ECoG signals (Lomtev, Kovalev, and Timchenko 2022) presents a convolutional encoder-decoder machine learning model with impressive results, the predicted movements having an average Pearson correlation of 0.67, which is the highest correlation currently published.
It was selected for this thesis due to state-of-the-art performance, being a relatively traditional neural network, and due to the paper's code being publicly available.
For preprocessing the ECoG data is first standardized and has it's mean subtracted for each channel. Then all activity outside 40-300 Hz is filtered out with a band-pass filter. The power grid frequency at 50 Hz and it's harmonics are also removed. Each channel is then convolved with complex Morlet wavelets to turn it into a time–frequency representation (TFR) of which only the real value (representing the amplitude) was kept. The sampling rate was then decimated from 1000 Hz to 100 Hz and the final output was rescaled using the RobustScaler from the scikit-learn library to crop outliers in the 0.1 and 0.9 quantiles. Finally the whole signal is shifted 200 ms to account for the delay between the brainwaves measured by ECoG and the physical finger flexions.
The preprocessing of the finger flexion data is much simpler, simply upscaling the data from 25 Hz to 100 Hz using bicubic interpolation and rescaling the result with a MinMax scaler.
The model architecture is that of a convolutional autoencoder, with minor refinements. The model takes an arbitrary length window of ECoG features and returns the predicted movements over that length of time. The first layer is a standard convolution layer to reduce the dimension. The layers after that are convolutional encoder layers, each of which consists of 1D convolution, normalization, GeLU activation, and max pooling with stride 2. This architecture is inspired by wav2vec (Schneider et al. 2019). A symmetrical set of decoder layers then follows. These layers are, in addition to their connection to the previous layer, also connected to their counterparts with a 'skip connection' inspired by UNet (Ronneberger, Fischer, and Brox 2015). The final layer is a 1x1 convolution layer to translate the resulting features into finger positions for each finger for each point in time.

Chapter 4

Methodology

The techniques discussed in this thesis are compared on several attributes to determine how well they perform when pretrained on several subjects and then introduced to a new subject.

The following steps are taken to perform measurements:
First, the technique is adapted for usage with the Stanford Fingerflex dataset (Miller 2019) if the original paper uses another dataset. This dataset contains 9 test subjects with a variety of electrode positions and counts. We decided to exclude two of these subjects because their recordings were taken on the opposite side of the brain from the other subjects. We split this dataset into several subsets each consisting of 6 subjects with one subject left out.

The models are then trained on these six subjects and a baseline comparison model is trained on the subject left out. Then, the models trained on the multi-subject data are tested on the subject left out, this is used to evaluate how well a given technique generalities to new subjects without fine-tuning. Afterwards, these pretrained models are then trained further on the new test subject.

Since the methods of training differs extensively between techniques, and much time could be spent researching better finetuning methods for these techniques, we opt to only implement the simplest naive method of finetuning available. Similar choices are made in data preparation, since most models evaluated expect a consistent number of channels in the training data we crop the channels to the lowest amount in our dataset. This means only the first 43 channels are used for each subject.

In the FingerFlex preprocessing pipeline the data needed to be processed in chunks as our equipment would otherwise run out of memory. This chunked preprocessing is kept for all other methods to ensure uniformity in our testing process.

4.1 FingerFlex

4.1.1 Introduction

The FingerFlex model is the first model we decided to evaluate. It is the single highest performing model currently published at an average Pearson correlation of 0.67. The structure of the model is easy to understand and it's code has been published online which sped up the evaluation process immensely.

4.1.2 Preprocessing

The ECoG preprocessing pipeline starts with normalizing the data. First the mean is subtracted for each channel, then the channel values are divided by their standard deviation. Then Common Average Referencing(CAR) is applied.

The next step is filtering the data. A band-pass filter is applied to filter out all activity outside 40-300 Hz since this range represents the gamma and high gamma components of the local field potential signal, which were chosen as the main predictors for the task. Noise created by the powerline at 50 Hz and it's harmonics was also removed using a notch filter.

In order to extract the temporal aspect of the data, the signal was convolved with complex Morlet wavelets. Forty wavelet kernels were created with base frequencies ranging from 40 to 300 Hz evenly spaced on a logarithmic scale. The amplitudes of the resulting signals were then extracted as the absolute value of the complex-valued result of the convolution operation. The resulting spectrograms were then downsampled by decimation from 1000 Hz to 100 Hz.

Finally, the ECoG data is rescaled using the scikit-learn RobustScaler with a quantile range of (0.1, 0.9). And to account for the delay between brain activity and actual movement, a forwards shift of 200 ms of the ECoG data respective to the finger flexion data was applied. Figure 4.2 shows how the ECoG data looks after preprocessing is done, each electrode's signal is split into forty channels and this image shows a couple channels sampled from the first electrode of subject bp.

The finger flexion preprocessing pipeline consists of only two steps. First the data is upsampled from 25 Hz to 100 Hz using bicubic interpolation. Second, the data is rescaled using the scikit-learn MinMax scaler into the $[0, 1]$ range.

4.1.3 Model

The model architecture is designed similarly to a convolutional autoencoder, with a couple of additions. The architecture of the model is depicted in fig. 4.3.

The model takes an arbitrary length window of ECoG features and returns the corresponding window of predicted finger flexions. During training the length of this window is fixed and set to 256 samples which corresponds to 2.56 seconds. The first layer is a regular convolution layer to reduce the dimension. This convolution layer uses the same structure as the encoder layers, which is: one-dimensional convolution, layer normalization, Gaussian Error Linear Units (GELU) activation, dropout, and finally one-dimensional max pooling. This is followed by a set of convolutional encoder layers, this part of the model is responsible for encoding the features of the time window fragments. This architecture is based on Schneider et al. 2019.

A symmetrical set of decoder layers then follows. These decoder layers differ from the encoder layers in only two respects, they use linear upsampling instead of maxpooling to downsample, and in addition to their connection to the previous layer they are also connected to their counterpart encoding layer. This 'skip connection' inspired by UNet (Ronneberger, Fischer, and Brox 2015) allows the decoding layer to use both latent space features and the original local features. The final layer is a 1x1 convolution layer to translate the resulting features into finger flexions for each finger for each point in time.

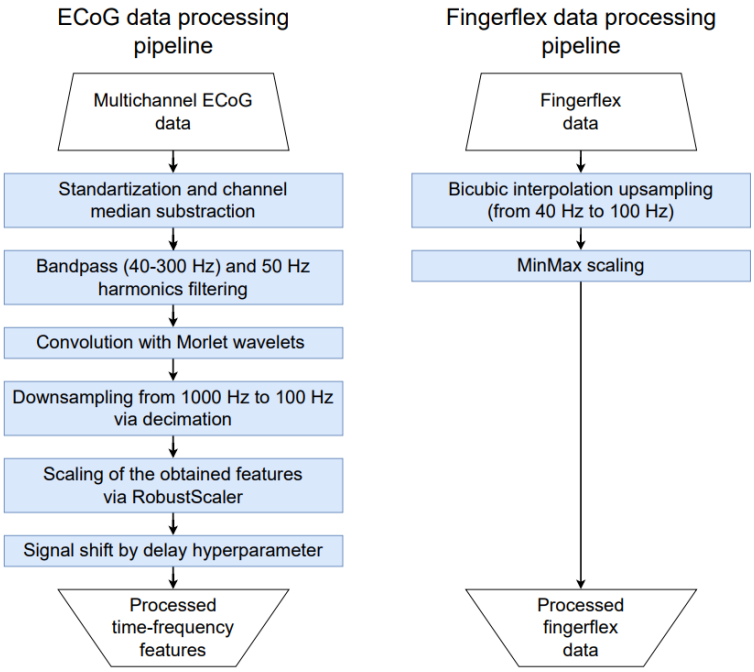


Figure 4.1: ECoG and finger motion data processing pipeline

Source: Lomtev, Kovalev, and Timchenko 2022

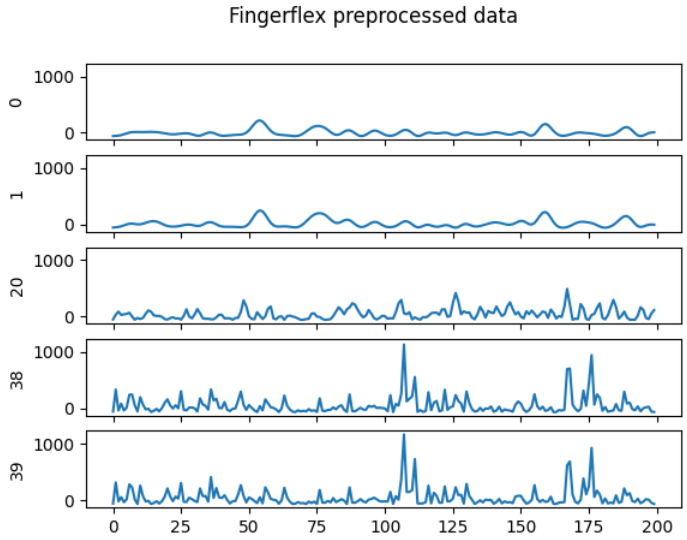


Figure 4.2: Example of processed ECoG data

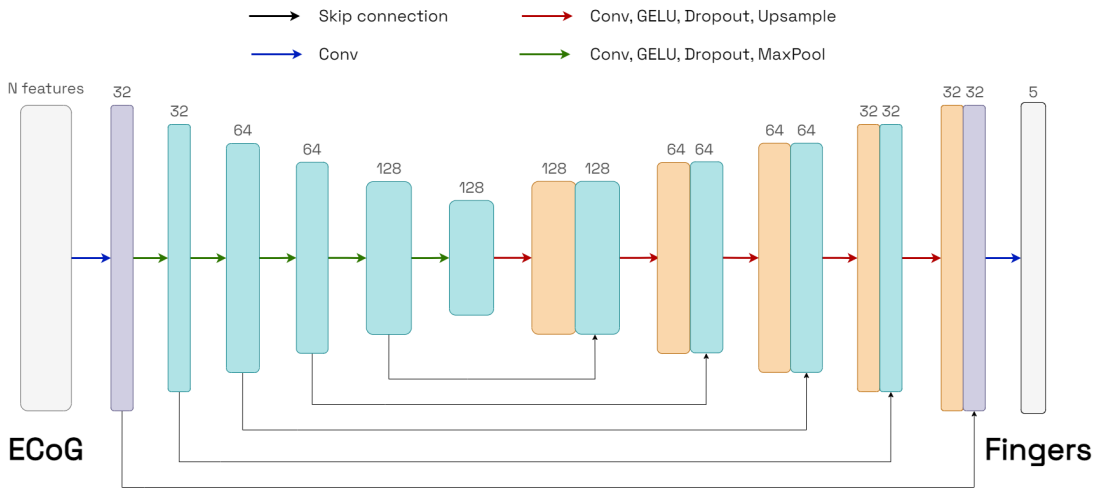


Figure 4.3: FingerFlex model architecture

Source: Lomtev, Kovalev, and Timchenko 2022

4.1.4 Modifications

Since the paper's code was publicly available there was little difficulty recreating the expected results. Though it should be noted that due to the variety of electrode amounts per subject in the dataset, some subjects have lesser performance than ideal because the data was cropped to only the first 43 channels for all subjects.

Additionally, due to memory limitations, the preprocessing step is applied to the data in chunks of 5 minutes (and a minimum of 10 seconds). Other than that there are no changes to the code. The algorithm used for this is described in fig. 4.4.

4.1.5 Training

Train-Test split The paper uses the same training split as provided by the BCI competition dataset, which used the first 6.5 minutes of the total 10 minutes of data as training data and the remaining 3.5 minutes as test data.

However since the Stanford dataset does not provide this split, and since the preprocessing needed to be split into chunks, the train-test split for this work instead uses the last 20% of each 5 minute chunk as the test set. For a single subject, since the recording is slightly longer than 10 minutes, this splits the data into two 4 minute training segments and two one minute test segments with a final 8 second training and 2 second test segment.

Loss function The loss function is the half sum of the mean squared error and the mean cosine distance. This combination was used as it was found to perform better than each metric alone. Model performance is still reported using the Pearson correlation coefficient, but this metric is too sensitive in small time windows to use as an optimization metric.

Hyperparameters The number of features in each layer is symmetrical to the bottleneck and set to (32, 32, 64, 64, 128, 128). The stride is set to 2 as this reduces the sampling rate as gradually as possible. The learning rate is reported in the paper as 8.4×10^{-5} though the source code provided uses 8.42×10^{-5}, the second number has been used for the experiments in this work. The reported learning rate was found using the Pytorch Lightning library's automatic learning rate finder. The amount of wavelets has been set to 40 and the time delay between the ECoG and finger flexion data has been set to 200 ms.

```
def chunk_data(data, chunk_size=300000, min_chunk_size=10000,
test_ratio=0.2, n_channels=43):
    #crop data to n_channels amount of channels
    data = crop(data, n_channels)

    #split into chunks
    for i in range(0, len(data), chunk_size):
        this_chunk_size = len(data[i:i + chunk_size])
        train_size = int(this_chunk_size * (1 - test_ratio))

        if this_chunk_size < min_chunk_size:
            continue
        result.append({
            'train_data': data[i:i + train_size],
            'test_data': data[i + train_size:i + chunk_size],
        })
    return result
```

Figure 4.4: pseudocode for splitting the data into chunks

4.1.6 Finetuning

Once a model checkpoint has been created which is trained on multiple subjects, this model is finetuned on a single subjects by loading the model weights using the Pytorch Lightning load_from_checkpoint function. The model is then trained for a further 20 epochs using the same parameters as the initial training method. It is likely that changing the hyperparameters for the finetuning step would improve performance, but due to the computational cost of training these models, multiplied by the need to train multiple models on larger datasets, we have not had the opportunity to experiment.

4.2 BTTR

4.2.1 Introduction

BTTR is the second model we chose to evaluate, primarily because it is a fundamentally different approach to the problem and would give a good idea as to how much our method would generalize. Additionally the mentor for this work is also one of the authors of this paper and was willing to help answer questions and provide a copy of the algorithm for us to use. Unfortunately this implementation contained a bug we could not discover, resulting in significantly worse performance than expected.

BTTR works iteratively to decompose the tensor into pruned sparse subdimensions, with each iteration correcting the error of the previous iterations. Compared to standard regression this reduces the amount of parameters and instead of starting the weighting from scratch each component is already correlated to the output.

4.2.2 Preprocessing

The finger flexion data is first, independently for each finger, normalized using z-score.

The ECoG data channels are re-referenced using Common Average Referencing (Ludwig et al. 2009). The paper then subjects the ECoG data to 8 bidirectional fourth-order Butterworth band-pass filters to split each channel into eight subchannels corresponding to the following frequency bands: δ(1.5 -5 Hz), θ(5 -8 Hz), α(8 -12Hz), β_1(12 -24 Hz), β_2(24 -34 Hz), γ_1(34 -60 Hz), γ_2(60 - 100 Hz), and γ_3(100 - 130 Hz). Certain frequency bands are known to correlate to certain types of brain activity.

For each cued finger flexion trial, the glove data and the bandpass filtered ECoG signals were extracted starting 1 sec prior to trial onset ("epoch"). This cuts out all non-movement sections besides the one second lead-up to the movement. ECoG epochs were downsampled to 10 Hz to further reduce data size. All epochs were then concatenated into a unique fourth-order tensor $\underline{\mathbf{X}} \in \mathbb{R}^{\text{Samples} \times \text{Channels} \times 8(\text{Frequencies}) \times 10(\text{Time})}$.

Finally the ECoG data is normalized using z-scores to reduce the difference in magnitude between frequency bands. The z-score of a value is the number of standard deviations that value is below or above the mean. A z-score is calculated as $z = \frac{x-\mu}{\sigma}$, with x the original value, μ the mean and σ the standard deviation.

This process is visualized in fig. 4.5.

4.2.3 Modifications

Originally the ECoG data was also manually inspected for bad channels to exclude, this manual filtering has not been performed for our measurements.

Due to the unknown bug in our implementation, some of the preprocessing steps resulted in even lower performance. We eventually decided to remove these steps to maximize performance despite the bug. The removed steps include the subdivision into different frequency bands, and the "epoch" extraction.

4.2.4 Algorithm

BTTR is based on Block Tensor Decomposition (De Lathauwer and Nion 2008). Tensor decomposition algorithms aim to decompose high-dimensional tensors into a sum of low-dimensional tensors containing the important information. An important consideration with tensor decomposition is deciding the rank of the tensors, too low and the tensors will be unable to fully represent complex data, too high and the tensor contains too much uninteresting information.

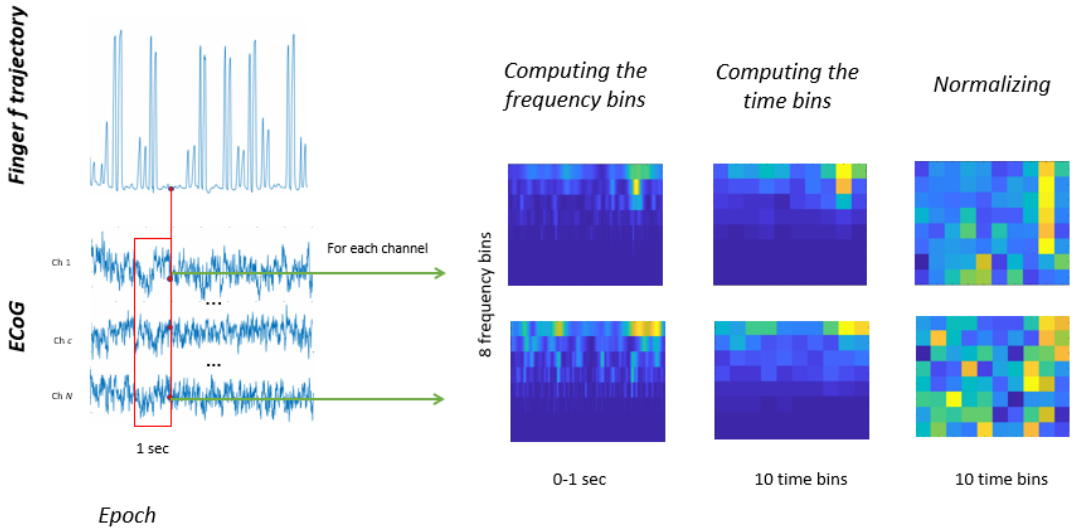


Figure 4.5: BTTR data preprocessing visualization

Source: Faes, Camarrone, and Van Hulle 2024

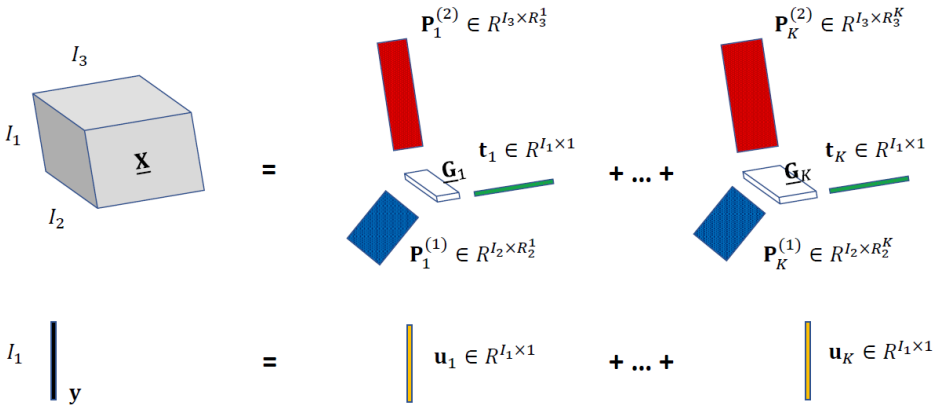


Figure 4.6: Scheme of BTTR algorithm with 3rd-order predictor variable $\underline{\mathbf{X}}$ and 1st-order response variable y.

Source: Faes, Camarrone, and Van Hulle 2024

Table 4.1: Mathematical notation

Notation	Description
$\underline{\mathbf{T}}, \mathbf{M}, \mathbf{v}, S$	tensor, matrix, vector, scalar (respectively)
$\mathbf{M}^T$	transpose of matrix
$\times_n$	mode-n product between tensor and matrix
$\otimes$	Kronecker product
$\circ$	outer product
$\|\cdot\|_F$	Frobenius norm
$\mathbf{T}_{(n)}$	mode-n unfolding of tensor $\underline{\mathbf{T}}$
$\underline{\mathbf{C}}^{(T)}$	core tensor associated to tensor $\underline{\mathbf{T}}$
$\mathbf{M}^{(n)}$	mode-n factor matrix
$\mathbf{M}_{ind}$	(sub-)matrix including the column(s) indicated in *ind*
$\mathbf{M}_{\setminus ind}$	(sub-)matrix excluding the column(s) indicated in *ind*
$[\![\underline{\mathbf{C}}; \mathbf{M}^{(1)}, ..., \mathbf{M}^{(N)}]\!]$	full multilinear product $\underline{\mathbf{C}} \times_1 \mathbf{M}^{(1)} \times_2 \cdots \times_N \mathbf{M}^{(N)}$
$\langle \underline{\mathbf{T}}, \underline{\mathbf{E}} \rangle_{\{n,n\}}$	mode-n cross-covariance tensor

The paper therefore proposes two methods to automatically choose an appropriate rank, Automatic Component Extraction (ACE), and its improved version augmented with automatic latent component selection, called Automatic Correlated Component Selection (ACCoS). The ACE and ACCoS models are embedded in the BTTR process to iteratively extract components that maximize the correlation between the multiway response and prediction. This thesis will focus on ACE-BTTR and all recorded data uses only ACE-BTTR.

Table 4.1 contains the mathematical notation used in this section to detail the algorithms BTTR uses.

Component Extraction

This section explains the algorithm BTTR uses to automatically extract component tensors which when summed together contain most of the information of the original input tensor. These components are selected to be maximally correlated between the input tensor $\underline{\mathbf{X}}$ and the output tensor $\mathbf{y}$.

The BTTR algorithm first extracts component tensors of the input, and in the next iteration extracts the component tensors of the difference between the input and and the previous iterations, up until the k'th iteration or until the difference is sufficiently small. As such each iteration of BTTR performs component extraction independently, which decomposes the tensor at that iteration into a sparse core tensor and several factor matrices which, when multiplied together reconstruct the original tensor. Each of these factor matrices correspond to the importance of features from a specific input dimension, with a core tensor containing the relations between components. This algorithm is described in detail in fig. 4.7.

In order to optimize for maximum sparsity and remove irrelevant components, a Bayesian information criterion (BIC) formula is used to select optimal parameters for the previous algorithm, the modified algorithm including this automatic parameter selection is referred to as ACE, described in fig. 4.8.

Input: $\underline{\mathbf{X}} \in \mathbb{R}^{I_1 \times \ldots \times I_N}, \mathbf{y} \in \mathbb{R}^{I_1 \times 1}, \tau$,*SNR*
Output: $\underline{\mathbf{G}} \in \mathbb{R}^{1 \times R_2 \times \ldots \times R_N}, \{\mathbf{P}^{(n)}\}_{n=2}^{N}$
Initialisation :
$\underline{\mathbf{C}} = \langle \underline{\mathbf{X}}, \mathbf{y} \rangle_{(1)} \in \mathbb{R}^{1 \times I_2 \times \ldots \times I_N}$
Initialisation of $\{\mathbf{P}^{(n)}\}_{n=2}^{N}$ and $\underline{\mathbf{G}}$ using HOOI on $\underline{\mathbf{C}}$
LOOP Process
repeat
  update $\underline{\mathbf{G}}$ using *SNR*
  prune $\{\mathbf{P}^{(n)}\}_{n=2}^{N}$ and $\underline{\mathbf{G}}$ using τ
until convergence is reached
return $\underline{\mathbf{G}}, \{\mathbf{P}^{(n)}\}_{n=2}^{N}$

Figure 4.7: Modified PSTD for component extraction

The following paragraphs provide a more in-depth explanation of the algorithm, and follow closely to their counterparts in Faes, Camarrone, and Van Hulle 2024.

Yokota and Cichocki 2014 proposes a method of pruning sparse Tucker decomposition (PSTD) of which the objective is to minimize the L1-norm of the core tensor under conditions of error bound and orthogonality constraints of individual basis matrices. At each iteration, the factor matrices and the sparsity of the core tensor are updated, and unnecessary dimensions removed according to the entries of the latter.

BTTR component extraction starts with a modification of PSTD that will be referred to as mPSTD. The mPSTD model is first initialized with HOSVD or HOOI[1], which are mostly interchangeable.

Iteratively we prune irrelevant components and increase sparsity using the soft-threshold parameter λ and the threshold $\tau \in [0, 100]$. Yokota and Cichocki 2014 finds the optimal sparsity λ with a line search of the signal-to-noise ration (SNR $\in [1, 50]$). The core tensor $\underline{\mathbf{G}}$ is updated using the soft-thresholding rule $\underline{\mathbf{G}} = sgn(\underline{\mathbf{G}}) \times max\{|\underline{\mathbf{G}}| - \lambda, 0\}$. The n-mode has irrelevant components removed using the formula $S^{(n)} = \{r | 100(1 - \frac{\sum_i \mathbf{G}_{(n)(r,i)}}{\sum_{t,i} \mathbf{G}_{(n)(t,i)}}) \geq \tau\}$, $\mathbf{P}^{(n)} = \mathbf{P}^{(n)}(:, S^{(n)})$ and $\mathbf{G}^{(n)} = \mathbf{G}^{(n)}(S^{(n)}, :)$

The mPTSD algorithm is summarized in Algorithm 4.7.

PTSD and mPTSD assume parameters such as the noise level (SNR) and component rejection threshold (τ) are known in advance. Naturally we would prefer to be able to derive the optimal value for these paremeters. BTTR provides automatic parameter selection for mPTSD using the following equations.

Defining the mode-1 cross-product between predictor and response variables as $\underline{\mathbf{C}} = \langle \underline{\mathbf{X}}, \mathbf{y} \rangle_{(1)} \in \mathbb{R}^{1 \times I_2 \times \ldots \times I_N}$ and its decomposition as $\underline{\mathbf{C}} \approx [\![\underline{\mathbf{G}}^{(c)}; \mathbf{P}^{(2)}, ..., \mathbf{P}^{(N)}]\!]$. Automatic parameter selection is calculated based on the Bayesian Information Criterion (BIC) defined as:

$$BIC(\tau, \text{SNR} | \text{SNR}, \tau^*) = \log\left(\frac{\|\underline{\mathbf{C}}\| - [\![\underline{\mathbf{G}}^{(c)}; \mathbf{P}^{(2)}, ..., \mathbf{P}^{(N)}]\!]_F}{s}\right) + \frac{\log(s)}{s} DF \tag{4.1}$$

with $\underline{\mathbf{G}}^{(c)}$ the sparse core matrix, and $\{\mathbf{P}^{(n)}\}_{n=2}^{N}$ the factor matrices, both obtained with mPSTD using specific τ and SNR values. s is the number of entries in $\underline{\mathbf{G}}$, and DF the degree

[1] Tucker Decompositions are usually not unique, so additional constraints are usually imposed such as orthogonality, sparsity, or non-negativity (De Lathauwer, De Moor, and Vandewalle 2000a; Zubair and Wang 2013; Phan and Cichocki 2008). Higher-Order Singular Value Decomposition (HOSVD) (De Lathauwer, De Moor, and Vandewalle 2000a) and Higher-Order Orthogonal Iteration (HOOI) (De Lathauwer, De Moor, and Vandewalle 2000b) are proposed variants of TKD with orthogonality constraints on the factor matrix.

Input: $\underline{\mathbf{X}} \in \mathbb{R}^{I_1 \times \ldots \times I_N}, \mathbf{y} \in \mathbb{R}^{I_1 \times 1}$
Output: $\underline{\mathbf{G}}^{(X)} \in \mathbb{R}^{1 \times R_2 \times \ldots \times R_N}, \mathbf{t}, \{\mathbf{P}^{(n)}\}_{n=2}^{N}$
$\underline{\mathbf{C}} = \langle \underline{\mathbf{X}}, \mathbf{y} \rangle_{(1)} \in \mathbb{R}^{1 \times I_2 \times \ldots \times I_N}$
Initialisation of $\tau = 90, ..., 100$; $SNR = 1, ..., 50$
for SNR_i in SNR **do**
  for τ_j in τ **do**
    $\underline{\mathbf{G}}, \{\mathbf{P}^{(n)}\}_{n=2}^{N} = mPSTD(\underline{\mathbf{X}}, \mathbf{y}, SNR_i, \tau_j)$
    calculate BIC value corresponding to SNR_i and τ_j using Eq 4.1
  end for
  select $\tau^* = \text{argmin}_\tau \text{BIC}(\tau)$
  calculate BIC value corresponding to SNR_i and τ^* using Eq 4.1
end for
select $SNR^* = \text{argmin}_{SNR} \text{BIC}(SNR, \tau^*)$
$\underline{\mathbf{G}}, \{\mathbf{P}^{(n)}\}_{n=2}^{N} = mPSTD(\underline{\mathbf{X}}, \mathbf{y}, SNR^*, \tau^*)$
$\mathbf{t} = (\underline{\mathbf{X}} \times_2 \mathbf{P}^{(2)T} \times_3 ... \times_N \mathbf{P}^{(N)T})_{(1)} vec(\underline{\mathbf{G}})$
$\mathbf{t} = \mathbf{t} / \|\mathbf{t}\|_F$
$\underline{\mathbf{G}}^{(X)} = [\![\underline{\mathbf{X}}; \mathbf{t}^T, \mathbf{P}^{(2)T}, ..., \mathbf{P}^{(N)T}]\!]$
return $\underline{\mathbf{G}}^{(X)}, \mathbf{t}, \{\mathbf{P}^{(n)}\}_{n=2}^{N}$

Figure 4.8: ACE

of freedom which is calculated as the number of non-zero elements in $\underline{\mathbf{G}}^{(c)}$, as suggested in Allen and Maletić-Savatić 2011.

Lower BIC values correspond to better candidates. For each SNR value, we can calculate the optimal τ as $\tau^* = \text{argmin}_\tau \, BIC(\tau, \text{SNR})$. Then, we can select the optimal SNR as $\text{SNR}^* = \text{argmin}_{\text{SNR}} \, BIC(\text{SNR}, \tau^*)$.

Once $\underline{\mathbf{G}}^{(c)}$ and $\{\mathbf{P}^{(n)}\}_{n=2}^{N}$ are computed, the score vector t is first calculated as

$$\mathbf{t} = (\underline{\mathbf{C}} \times_2 \mathbf{P}^{(2)T} \times_3 ... \times_N \mathbf{P}^{(n)T})_{(1)} \text{vec}(\underline{\mathbf{G}}^{(c)}),$$

and then normalized. Faes, Camarrone, and Van Hulle 2024 refers to this fully automatic component extraction as ACE, summarized in Algorithm 4.8.

Block-Term Tensor Regression (BTTR)

The regression part of BTTR uses linear regression where the regression coefficient is multiplied by a weight matrix consisting of the component tensors extracted using ACE. For example, the blue component in Figure 4.6 could represent the channels dimension of our original data and its most important channels, while the red component does the same for the frequency bands, etc. The core tensor $\underline{\mathbf{G}}_k$ at each iteration represents the relations between these components.

The following in-depth description of BTTR regression follows closely to the text of Faes, Camarrone, and Van Hulle 2024.

Given a set of training data $\underline{\mathbf{X}}_{\text{train}} \in \mathbb{R}^{I_1 \times \ldots \times I_N}$ and vectoral response $\mathbf{y}_{\text{train}} \in \mathbb{R}^{I_1}$, BTTR training consists of automatically identifying K blocks such that

$$\underline{\mathbf{X}}_{\text{train}} = \sum_{k=1}^{K} \underline{\mathbf{G}}_k \times_1 \mathbf{t}_k \times_2 \mathbf{P}_k^{(2)} \times_3 ... \times_N \mathbf{P}_k^{(n)} + \underline{\mathbf{E}}_k$$

$$\mathbf{y}_{\text{train}} = \sum_{k=1}^{K} \mathbf{u}_k + \mathbf{f}_k \text{ with } \mathbf{u}_k = \mathbf{t}_k b_k$$

Input: $\underline{\mathbf{X}} \in \mathbb{R}^{I_1 \times ... \times I_N}, \mathbf{y} \in \mathbb{R}^{I_1 \times 1}, K$
Output: $\{\mathbf{P}_k^{(n)}\}, \{\mathbf{t}_k\}, \underline{\mathbf{G}}_k^{(X)}$ for $k = 1, ..., K$; $n = 2, ..., N$
Initialisation of $\underline{\mathbf{E_1}} = \underline{\mathbf{X}}$ and $\mathbf{f_1} = \mathbf{y}$
for $k = 1$ to K **do**
  if $\|\underline{\mathbf{E_k}}\| > \epsilon$ and $\|\mathbf{f_k}\| > \epsilon$ **then**
    $\underline{\mathbf{C_k}} = \langle \underline{\mathbf{E}}_k, \mathbf{f}_k \rangle_{\{1,1\}}$
    $\underline{\mathbf{G}}_k^{(X)}, \mathbf{t}_k, \mathbf{P}_k^{(2)}, ..., \mathbf{P}_k^{(N)} = ACE(\underline{\mathbf{E}}_k, \mathbf{f}_k)$ or $ACCoS(\underline{\mathbf{E}}_k, \mathbf{f}_k)$
    $\mathbf{b_k} = \mathbf{t_k f_k}$
    $\underline{\mathbf{E_{k+1}}} = \underline{\mathbf{E_k}} - [\![\underline{\mathbf{G}}_k^{(X)}; \mathbf{t}_k, \mathbf{P}_k^{(2)}, ..., \mathbf{P}_k^{(N)}]\!]$
    $\mathbf{f_{k+1}} = \mathbf{f_k} - \mathbf{t_k b_k}$
  else
    break
  end if
end for

Figure 4.9: BTTR

where $\underline{\mathbf{G}}_k \in \mathbb{R}^{1 \times R_2^k \times ... \times R_N^k}$ is the core tensor for the kth-block, $\mathbf{P}_k^{(n)}$ the kth loading matrix for the n-mode, $\mathbf{u}_k$ and $\mathbf{t}_k$ the score vectors, b_k the regression coefficient, and $\underline{\mathbf{E}}_k$ and $\mathbf{f}_k$ the residuals.

Once the model is trained and, as a consequence, $\underline{\mathbf{G}}_k$, $\mathbf{P}_k^{(n)}$ and b_k are computed. We can obtain the final prediction as: $\mathbf{y}_{\text{test}} = \mathbf{Tb} = \mathbf{X}_{\text{test}(1)}\mathbf{Wb}$ where each column $\mathbf{w}_k = (\mathbf{P}_k^{(n)} \otimes ... \otimes \mathbf{P}_k^{(2)}) vec(\underline{\mathbf{G}}_k)$.

A visual representation is shown in fig. 4.6 while the full process is described in Algorithm 4.2

4.2.5 Training

The Train-Test split is the same as discussed in FingerFlex (section 4.1.5) for consistency, and the training method involves applying the algorithm described in the previous section to the training data. A K value of 100 was used, since this was sufficient to trigger early stopping in the algorithm and we saw no sign of overfitting.

4.2.6 Finetuning

The BTTR algorithm does not lend itself towards finetuning, as each iteration depends on the residual of the previous step there is no way to insert new data into the training process without significant changes to the algorithm.

The 'finetune' column is therefore replaced with an 'extended' column. This column measures performance when the model is trained on a dataset containing every subject, including the same subject the test data is from.

Training on a multi-subject dataset is done by creating a multi-subject tensor through concatenating each individually preprocessed chunk of data into a single tensor along the temporal axis.

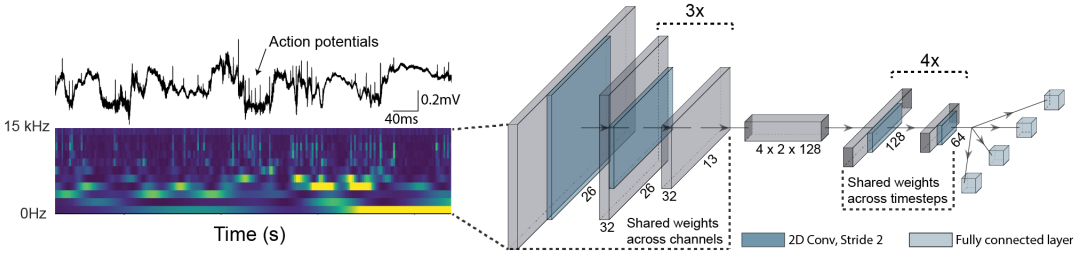


Figure 4.10: DeepInsight model architecture

Source: Frey et al. 2021

4.3 DeepInsight

4.3.1 Introduction

DeepInsight(Frey et al. 2021) is a convolutional neural network for decoding neural data designed to generalize across a wide variety of brain regions, behaviors, and recording techniques. It's main purpose is to allow neuroscientists to apply the model to unprocessed data and examine the resulting weights to identify relevant brain regions, channels, or frequencies. The paper applies the model to decode the spatial position of freely roaming rats based on signals recorded from the hippocampal region CA1, but the ECoG data from the BCI competition is used to showcase how the model architecture generalizes across wildly different datasets. In this work we will focus on the parts of the paper that apply to the BCI competition dataset.

This model was selected for evaluation in this work because it is designed to generalize well and intended to require little preprocessing. It is also relatively similar to the Fingerflex model and comparing both may be informative.

4.3.2 Preprocessing

Raw ECoG recordings are transformed into a frequency representation using a discrete wavelet transform (DWT) using morlet wavelets. Specifically the morlet wavelet defined as

$$\psi_0(\eta) = \pi^{\frac{1}{4}} * \exp(i * \omega_0 * \eta) * \exp(-\eta^2/2) \tag{4.2}$$

with ω_0 the non-dimensional frequency constant equal to 6.

These wavelets are then downscaled by a factor M using averaging, which is $M = 1000$ in the default model as the CA1 data has a sampling rate of 30000 Hz. For our ECoG data the paper uses a downscaling factor of 50.

Finally they apply normalization using median absolute deviation (MAD) to both the frequencies and channels of the input data using the following formula:

$$X_{cf} = \frac{X_{cf} - \tilde{X}_{cf}}{\text{median}(|X_{cf} - \tilde{X}_{cf}|)} \tag{4.3}$$

where $\tilde{X}_i$ is the median of X_i.

4.3.3 Model

The model takes an input in the shape (batch_size, channels, timesteps, frequencies) and applies gaussian noise $\mathcal{N}(\mu, \sigma^{\in})$ with $\mu = 0$ and $\sigma = 1$. This is then followed by eight convolution layers. These layers use a kernel size of 3 and 64 filters. The layers share weights on the

channel axis, which greatly reduces the amount of trainable parameters and increases decoding performance. With a stride of 2, these layers alternate between downsampling the time and frequency dimensions.

After these eight layers the channel dimension is reduced with a series of convolutional layers until only two channels remain. These layers have a stride of two, 128 filters, and share weights in the time dimension instead of the channel dimension.

Finally, two fully connected dense layers are appended to perform the final regression. In cases where multiple different outputs are required these final layers are separate for each output, but in our case only a single output is needed.

Figure 4.10 shows the entire model architecture with multiple 'regression heads' connected to the main model to decode multiple outputs.

4.3.4 Training

The model input shape consists of 43 channels, 128 timesteps (corresponding to 6.4 seconds), and 16 frequencies logarithmically scaled between 0 Hz and 15000 Hz.

The same train-test split as all other models is used, but the training data is additionally split into five cross-validation partitions. With the first partition being used for validation and partitions 2-5 used for training, then for the second run partitions 1 and 3-5 are used for training and partition 2 for validation, and so on. Gaps are used between partitions to avoid any overlap from using 6 second long samples. However, due to time constraints we could not afford to train this model five times, so only the first cross-validation split is trained.

The Adam optimizer is used with a learning rate of 0.0007. This learning rate is multiplied by 0.2 if validation performance did not increase for 3 epochs. An epoch here does not correspond to the full dataset, but instead consists of 250 training steps. With a batch size of 8 this corresponds to 2000 randomly sampled points from our dataset each 'epoch'.

The model is trained for 20 of these epochs.

4.3.5 Finetuning

The model weights are loaded and the exact same settings are used to further train the model on new data. The learning rate modification is not recorded so finetuning starts with a learning rate of 0.0007 and multiplies it by 0.2 on plateauing just as in normal training.

4.4 HistGradientBoosting

4.4.1 Introduction

For the final model to evaluate we decided to make use of the algorithms provided by the scikit-learn library (version 1.6.1) instead of recreating an existing paper. We tested a variety of models provided by the library and evaluate the highest performing of these according to the same metrics as the other models evaluates in this work. As many of the tested regressors do not provide methods to finetune the model, or do so in a variety of ways, it was not practical to do full multi-subject testing for each regressor.

As such, the model which was to be fully tested was selected based on it's single-subject performance on subject bp, as this subject consistently showed the best correlation across models. A variety of preprocessing steps were also explored, such as the morlet wavelet convolution used in FingerFlex (section 4.1) and the band-pass filtering of BTTR (section 4.2). The model described in the following section is the result of this experimentation.

4.4.2 Preprocessing

The finger and ECoG data is first shifted by 200 ms to account for the time delay between the brain signals and the actual measured movements. A shift of 200 ms has been experimentally found to yield better model performance.

Both are then normalized using z-scores, independently for each finger or electrode channel. The ECoG data channels are then re-referenced using Common Average Referencing. Both the finger flexion data and the ECoG data are then downsampled from 1000 Hz to 50 Hz and a two second sliding window of ECoG data is provided as input for each finger flexion data point to provide the model with enough information on the previous state. A higher sample rate and a larger window have been found to slightly increase model performance at the cost of significantly increasing the computational cost, and can be regarded as tunable hyperparameters along with the time delay shift.

As most scikit-learn models only support input in the shape (n_samples, n_features) the ECoG data is flattened from the shape (n_samples, window_size, n_channels) to the shape (n_samples, window_size * n_channels).

A threshold is then applied to the finger flexion data, setting the all flexions less than 30% of the maximum to zero, to reduce noise and remove small movements.

Finally, the finger flexion data is converted into two datasets, a Y_{max} and Y_{class} which respectively record the maximum flex of the fingers and the label of which finger has the max flexion. This last step is only applied to the training data in order to train the regressor and classifier model respectively. Preprocessing for the test data stops at the thresholding step.

4.4.3 Model

We tested a variety of regressors before focusing on the HistGradientBoostingRegressor which was experimentally found to perform the best on single subject data. Some of the other regressors we tested include: SVR, RandomForestRegressor, PassiveAggressiveRegressor, HuberRegressor, SGDRegressor, and AdaBoostRegressor. Table 5.4 is a table of the mean correlation of each method using the same pre- and postprocessing for subject bp. The SGDRegressor failed to converge, and the RandomForestRegressor test was shut down as the training function had not finished after ten hours. The HistGradientBoostingRegressor gave the best result.

We finally settled on a combination of a HistGradientBoostingRegressor and HistGradientBoostingClassifier as we found that using separate models to predict which finger is flexed and the amount of flex increased the overall correlation. This combination is included in table 5.4 as *HistGradientBoosting**.

```
for finger in range(5):
    pred = regressor.predict(X_test)
    pred *= classifier.predict_proba(X_test)[:, finger]
    pred = scipy.ndimage.gaussian_filter1d(pred, 6)
```

Figure 4.11: Code sample generating the predicted finger flexions

RandomForest is an ensemble model which trains a number of decision trees on subsamples of the dataset. In our case this defaults to 100 decision trees. For classification the output of the random forest is the class selected by the most trees. For regression it is the average of all trees.

The SDGRegressor class available in scikit-learn implements a simple linear model fitted using stochastic gradient descent on a given loss function. In our case we used ordinary least squares fit. The Passive-Aggressive regressor implements an algorithm for 'online' machine learning, where the data becomes available in sequential order and the model updates it's prediction as more data arrives.

The Huber regressor is similar to the SDGRegressor except it uses the Huber loss function, which is a variant of squared error loss. Huber loss is quadratic for small values and linear for larger values, which means it is less sensitive to outliers than squared error loss. Huber loss is defined in the following equation, with a the difference between the prediction and actual value, and δ an epsilon. In our case $\delta = 1.35$.

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \leq \delta, \\ \delta \cdot \left(|a| - \frac{1}{2}\delta\right), & \text{otherwise.} \end{cases} \tag{4.4}$$

SVR uses Support Vector Regression with a radial basis function kernel.

AdaBoost is an ensemble model that fits multiple weaker regressors on the dataset. The first iteration simply fits the regressor to the original dataset. With each further iteration the data is reweighted such that incorrectly predicted points are given more weight in order to focus the next regressor on these cases. A weighted sum of the output from the trained regressor is then used to calculate the final output. In our tests the base regressor to be boostes was a decision tree regressor with max depth of three.

HistGradientBoosting is a Gradient Boosted Tree model based on LightGBM (Ke et al. 2017) which bins sample values into integer-valued bins in order to improve training speed by using integer-based calculations. Gradient boosted trees use gradient boosting on decision trees, which means they are ensemble models which use a large amount of weak learners (decision trees) to create a strong learner. Each iteration new trees are trained on the difference between the current prediction and the true values, weighted based on the negative gradient of the loss function. As a result each new tree is focused on correcting the error in the previous prediction.

The regressor learns the magnitude of the flexion while the classifier identifies the correct finger being flexed. We use the predict_proba() method provided by scikit-learn to get the chance for each finger that it is moving at that time. The final prediction is calculated by multiplying the prediction of the regressor with the probabilistic prediction of the classifier for a given finger.

As a last step gaussian noise is added to the prediction with $\sigma = 6$ as shown in fig. 4.11

4.4.4 Training and Finetuning

The scikit-learn implementation did not provide a method to finetune the HistGradientBoosting models. However, a warm_start option is available, which lets one add more estimators to an already fitted model. Using this option we can train the model one iteration at a time and keep increasing the number of iterations.

We train each model for 100 iterations, which is the default number of iterations, and a sufficiently high number to allow the model to converge. For this training the we use the fit() method provided by scikit-learn, and default values for every parameter besides warm_start and the number of iterations.

In the finetuning step the multi-subject model is then trained for a further 100 iterations on the subject that was excluded, using the same method.

The Train-Test split is slightly different to that of the other models, since the sliding window during preprocessing would be larger than the length of the validation sample from the 10 second dataset chunk. As a result this third and last chunk has been left out of the dataset for this model by increasing the minimum chunk size parameter seen in fig. 4.4 to 20000.

Chapter 5

Results

5.1 FingerFlex

Results are promising, five out of seven of the subjects show a slightly higher performance from the finetuned model than the model trained from scratch on only that single subject. Unfortunately this gain in performance is relatively small, more interesting is the low amount of retraining the model requires. While the untuned general model does not perform well on a new subject, after only a single epoch of training it rapidly reaches the peak performance expected for the subject. We hypothesize that the need for finetuning could be even further reduced by standardizing electrode locations across subjects. The error bars present in the graphs represent the standard deviation of the validation correlation, as this varied slightly over multiple tests.

Figure 5.1 shows a sample of the various models predictions on subject bp. Each prediction was individually rescaled as it varies significantly and would otherwise make the graphs unreadable. We can clearly see that the general model has issues identifying the correct finger but does not usually spike for no reason. The model trained from scratch also has this issue, while the finetuned model is much more accurate and generally spikes less when other fingers are moving.

Figure 5.2 shows the average correlation over all subjects, and showcases a clear trend towards a slightly higher correlation and how significantly less training is required to reach a high correlation. Figure 5.3 and table 5.1 show an overview of each individual subject's final results and separates out the zeroth epoch from the others to show how the model performs with zero finetuning compared to after 20 epochs of training/finetuning. The performance of the general model when not finetuned is unreliable at best, though once finetuned consistently outperforms

Patient Code	From Scratch	Finetune	Untrained
bp	0.5766	0.6349	0.1771
zt	0.6200	0.5673	0.0549
jp	0.4921	0.5158	0.0188
ht	0.1608	0.1592	0.0070
mv	0.5290	0.5849	−0.0820
wc	0.2323	0.3020	0.0190
jc	0.4102	0.5007	−0.0455
average	0.4316	0.4664	0.0213

Table 5.1: FingerFlex mean correlation per subject

that of a model trained on only a single subject.

Subject bp as shown in fig. 5.4a has the highest performance of all subjects in each metric. The generalized model performs well, and only a single epoch of finetuning improves performance to the peak performance of the model trained from scratch. Overall the finetuned model consistently outperforms the model trained on only a single person.

Subject zt in fig. 5.4b shows that sometimes the generalized model performs worse even when finetuned. Figures 5.4c and 5.4d show near identical performance between the finetune and the single-subject model, though the finetune consistently requires less epochs of training.

Figure 5.4e shows the greatest difference in training time. The general model had a negative correlation before finetuning but after a single epoch immediately reaches its peak performance. The single-subject model needs four epochs of training to reach a similar correlation and still performs slightly worse.

Figures 5.4f and 5.4g interestingly reach a very similar performance between the generalized and single-subject model around epoch 5 of training, with the generalized model then proceeding to perform notably better than the single-subject model. Unfortunately we could not afford to train these models multiple times to ascertain if this is a consistent phenomenon or merely the finetune getting lucky while the other model didn't.

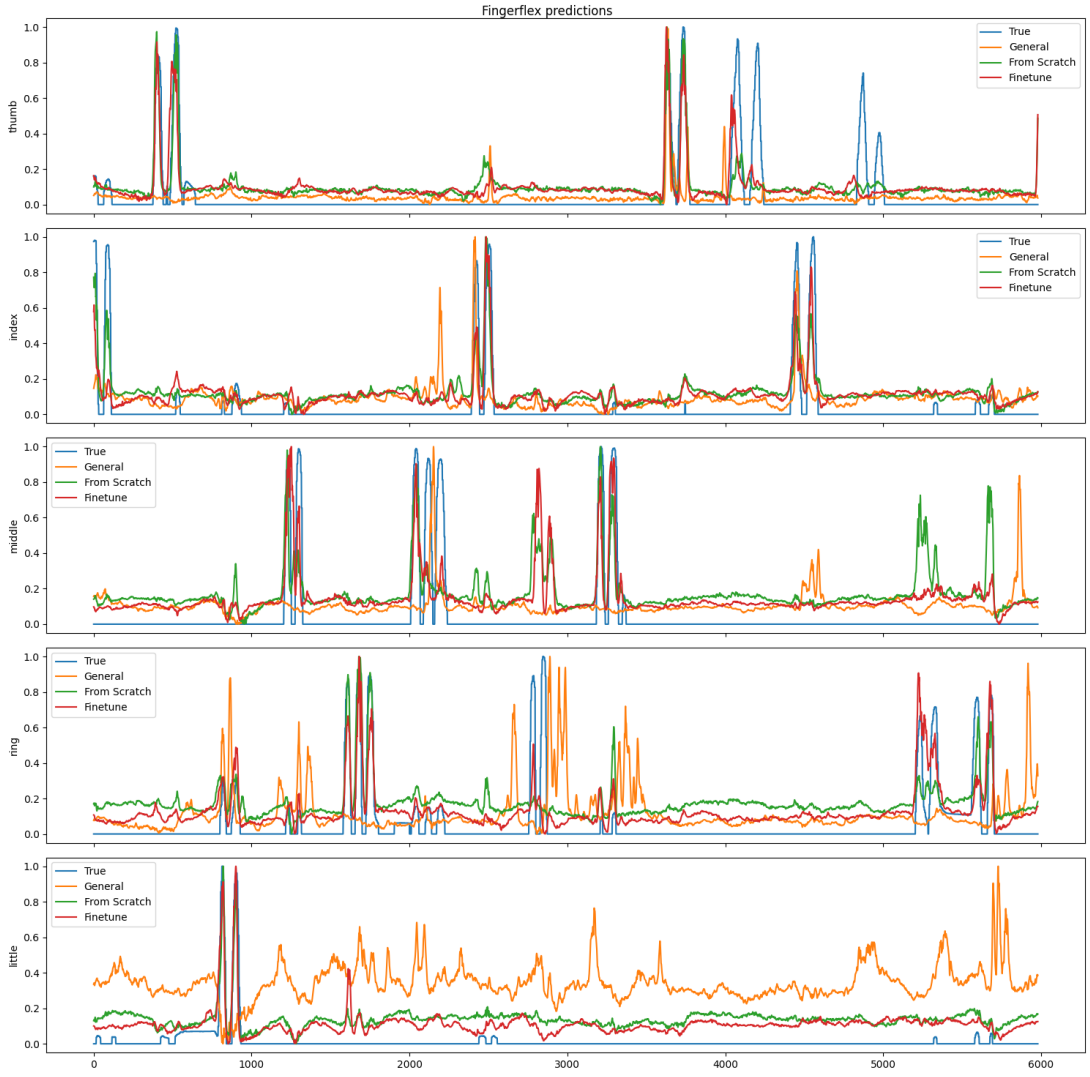


Figure 5.1: Fingerflex predictions on subject bp

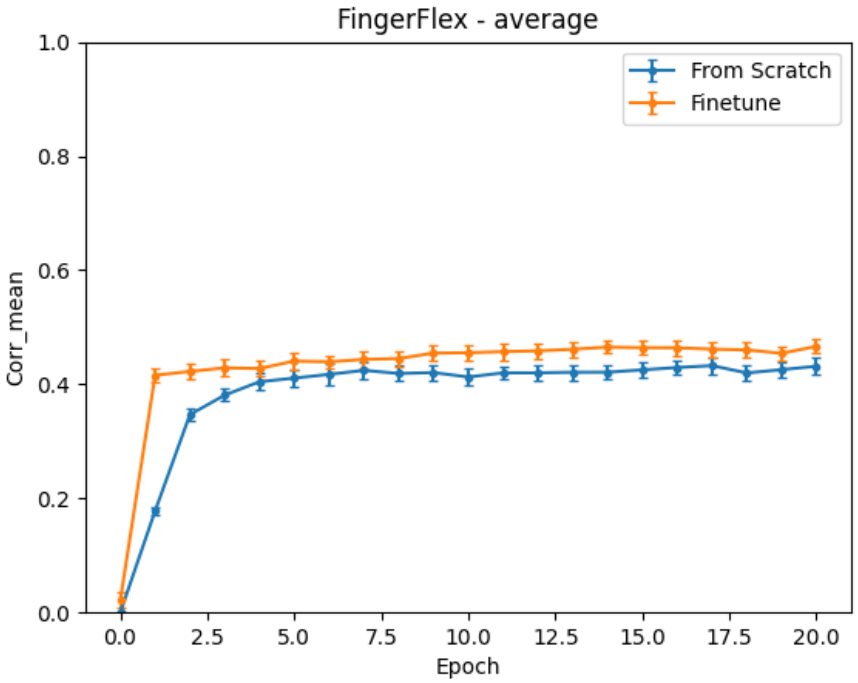


Figure 5.2: Average FingerFlex mean correlation over epochs

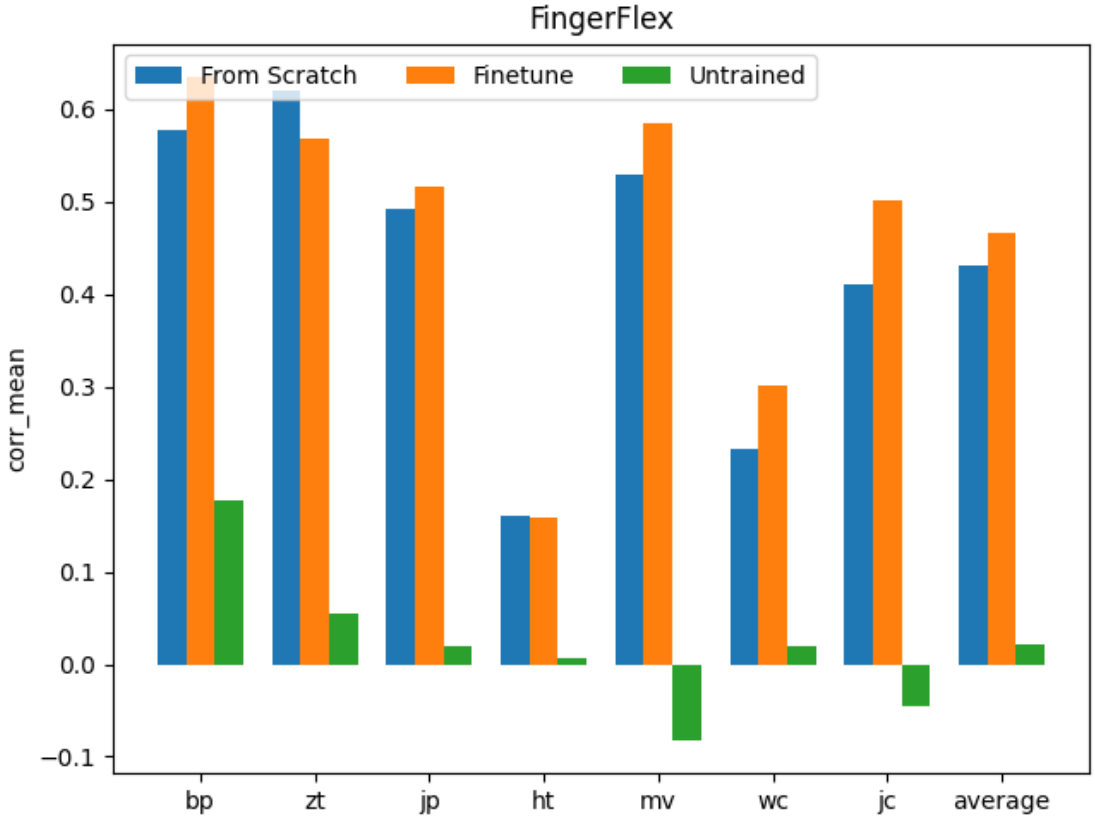
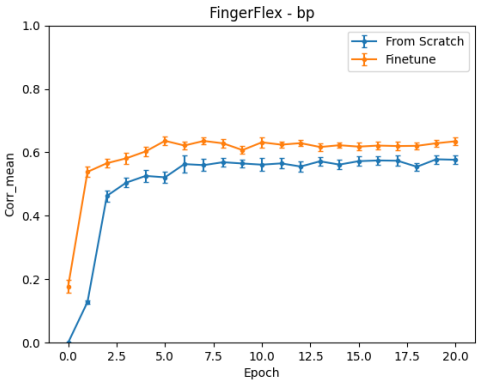
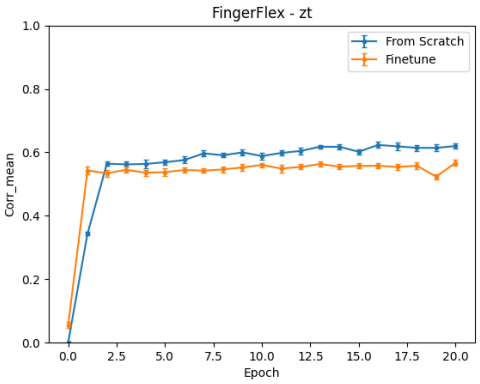


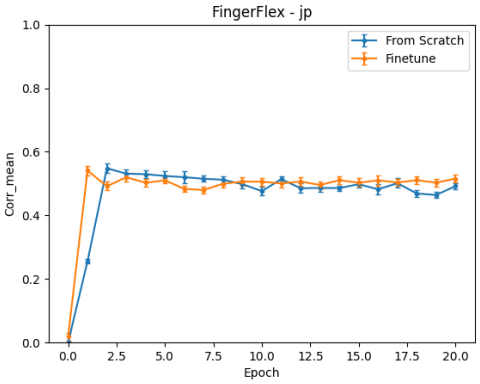
Figure 5.3: FingerFlex performance across subjects



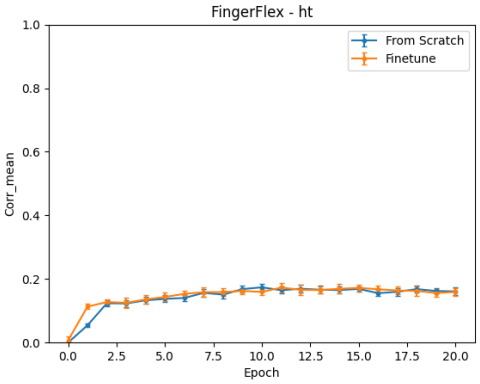
(a)



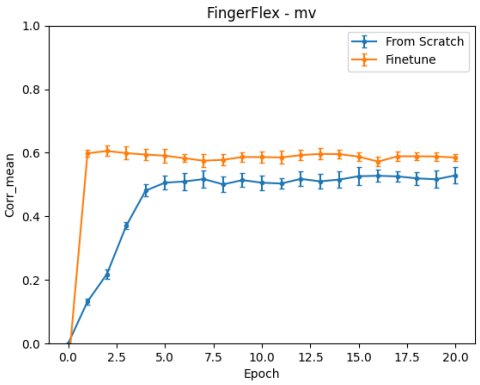
(b)



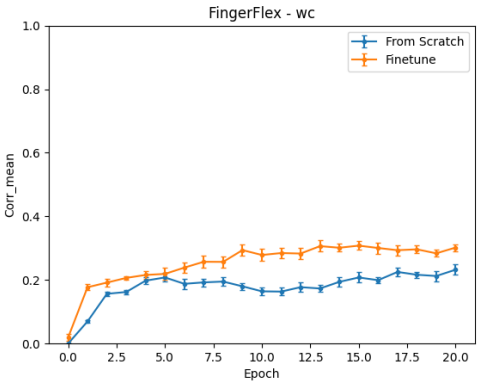
(c)



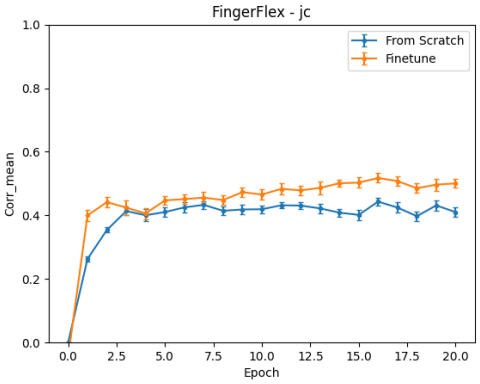
(d)



(e)



(f)



(g)

Figure 5.4: FingerFlex mean correlation across subjects over epochs

Patient Code	From Scratch	Extended	Untrained
bp	0.1029	0.0656	−0.0119
zt	0.0757	0.0307	−0.0285
jp	0.0302	0.0296	0.0114
ht	0.0880	0.0556	−0.0118
mv	0.1691	0.0083	−0.0371
wc	0.1162	0.0493	0.0112
jc	0.1919	0.0547	−0.0297
average	0.1106	0.0420	−0.0138

Table 5.2: BTTR mean correlation per subject

5.2 BTTR

Extending the dataset does not appear to have any positive effect on the performance of BTTR. In fact nearly every subject's model performs significantly better when trained on only that subject's data, as shown in fig. 5.7 and table 5.2. Performance on unseen subjects is even worse, the prediction barely better than pure random guessing.

Only on subject jp does the extended model even come close to the performance of the single-subject model, and that is likely only due to the extremely low performance of BTTR on this subject (due to an unknown bug in our implementation).

We believe that standardizing the ECoG data across subjects would help significantly for BTTR, as it does not have the flexibility of a neural network where a single layer can adapt to the difference in electrode positions without the deeper layers needing to change.

Figure 5.5 shows a sample of the various models predictions on subject bp. Each prediction was individually rescaled as it varies significantly and would otherwise make the graphs unreadable. It's clear that all models have significant issues decoding the finger flexions, leaving little to discuss.

Figure 5.6 shows the average correlation across all subjects. Correlation from performance on unseen subjects is practically nonexistent. We can see a clear trend where single-subject models perform better than those trained on multiple subjects.

Figure 5.7 displays the final values for each subject and the averages. Single-subject models consistently outperform those with extended datasets, who in turn tend to outperform a model trained only on other subjects.

Figure 5.8 has each individual subject plotted out, though most follow a similar pattern as seen in fig. 5.6. Figure 5.8f does show an interesting section around epochs 3-11, where the extended model has nearly the same accuracy as the single-subject model.

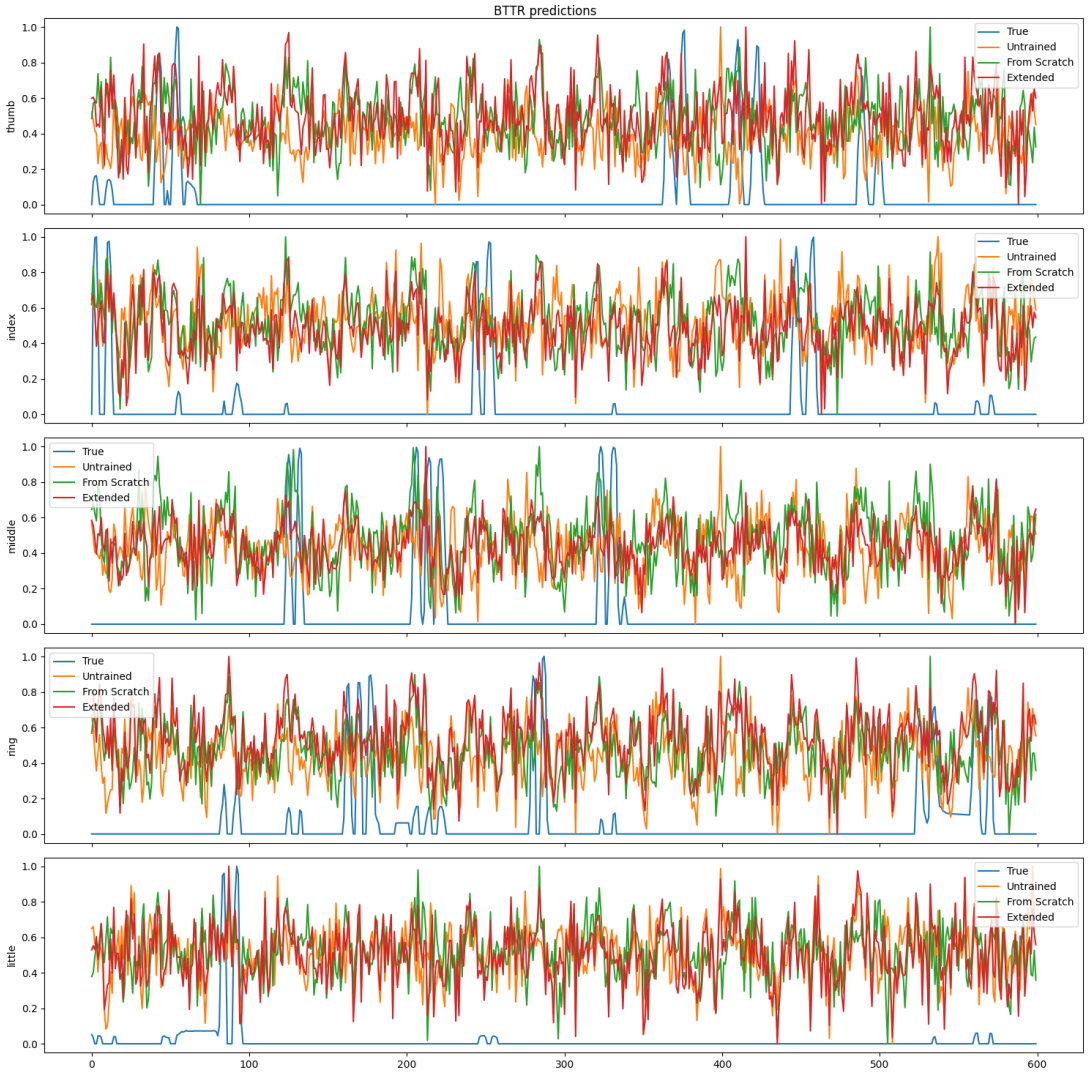


Figure 5.5: BTTR predictions on subject bp

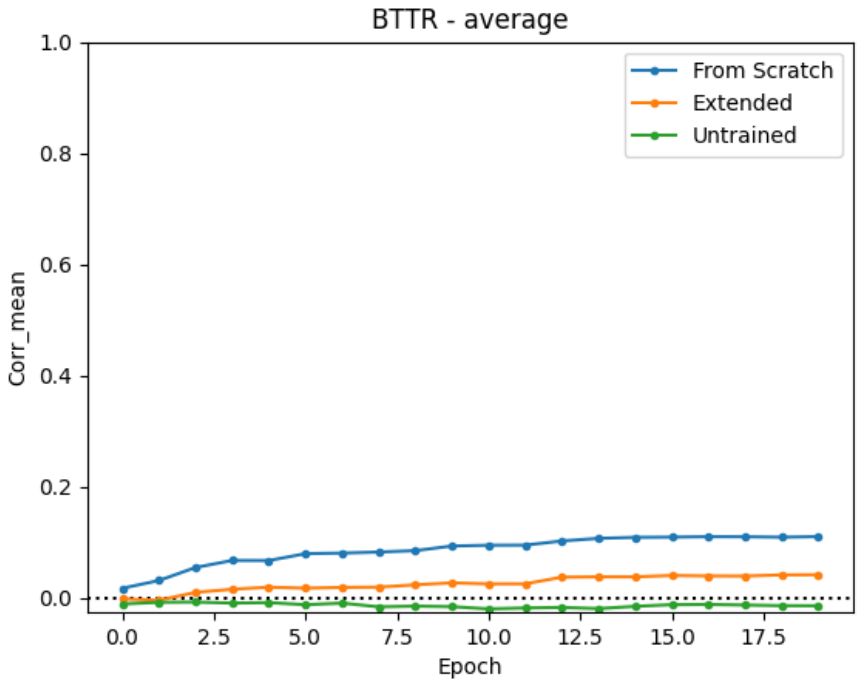


Figure 5.6: Average BTTR mean correlation over epochs

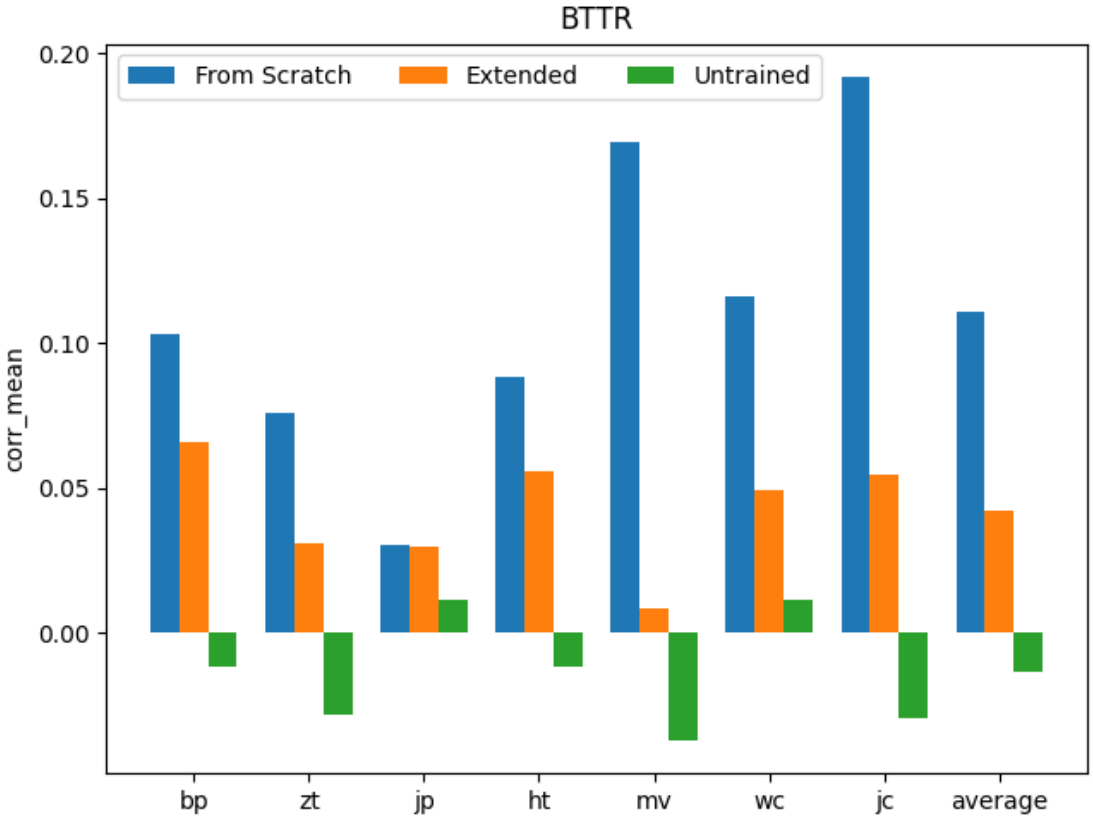


Figure 5.7: BTTR performance across subjects

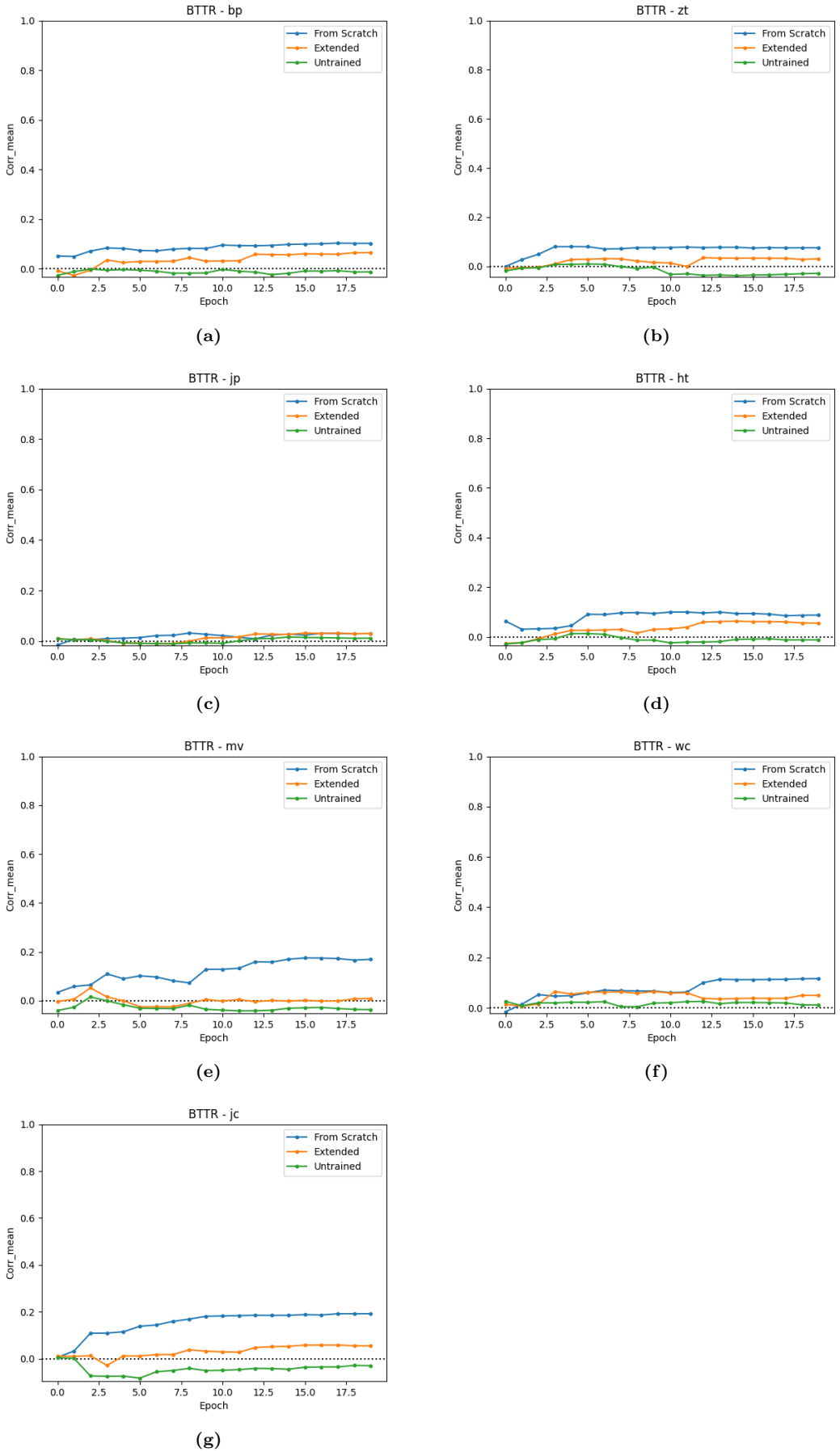


Figure 5.8: BTTR mean correlation across subjects over epochs

Patient Code	From Scratch	Extended	Untrained
bp	0.5658	−0.0356	−0.0391
zt	0.3753	−0.0265	0.0082
jp	0.1412	0.0012	0.0007
ht	0.0166	−0.0727	−0.0539
mv	0.1398	0.0004	−0.0111
wc	0.0145	0.0755	−0.0047
jc	0.5298	0.0258	0.1328
average	0.2547	−0.0046	0.0047

Table 5.3: DeepInsight mean correlation per subject

5.3 DeepInsight

DeepInsight has the worst multi-subject performance of all tested models. For every subject the model fails to achieve any notable correlation during finetuning, even moving closer to zero in the case of fig. 5.12g. We believe that the main difference between Fingerflex's performance and this model is that Fingerflex has an initial dense layer before the convolution layers. We theorize that during finetuning this layer rearranges the input electrode channels such that the rest of the model can maintain it's original weights with minimal adjustments. Essentially serving as a translation layer between subjects.

Figure 5.9 shows an individually rescaled sample of the various models predictions on subject bp. The model trained from scratch has a clear correlation to the true finger flexions, but both the general and finetuned model are nearly completely unrelated to the movements they are supposed to predict.

Figure 5.10 shows the average correlation across all subjects. Correlation from performance on unseen subjects is practically nonexistent, with the model clearly failing to predict the finger flexions at all.

Figure 5.11 displays the final values for each subject and the averages. Multi-subject training is clearly completely unreliable compared to single-subject training.

Figure 5.12 has each individual subject plotted out. With figs. 5.12d and 5.12f the model fails to converge during single-subject training, interestingly in fig. 5.12f the multi-subject finetuning actually surpasses the performance of the single-subject model.

Figures 5.12a to 5.12c, 5.12e and 5.12g all show a similar trend, normal convergence during single-subject training and a complete failure to predict finger flexions during finetuning. Interestingly fig. 5.12g actually had some correlation from the initial general model but finetuning appears to lower the performance instead of improve it.

Figure 5.9: DeepInsight predictions on subject bp

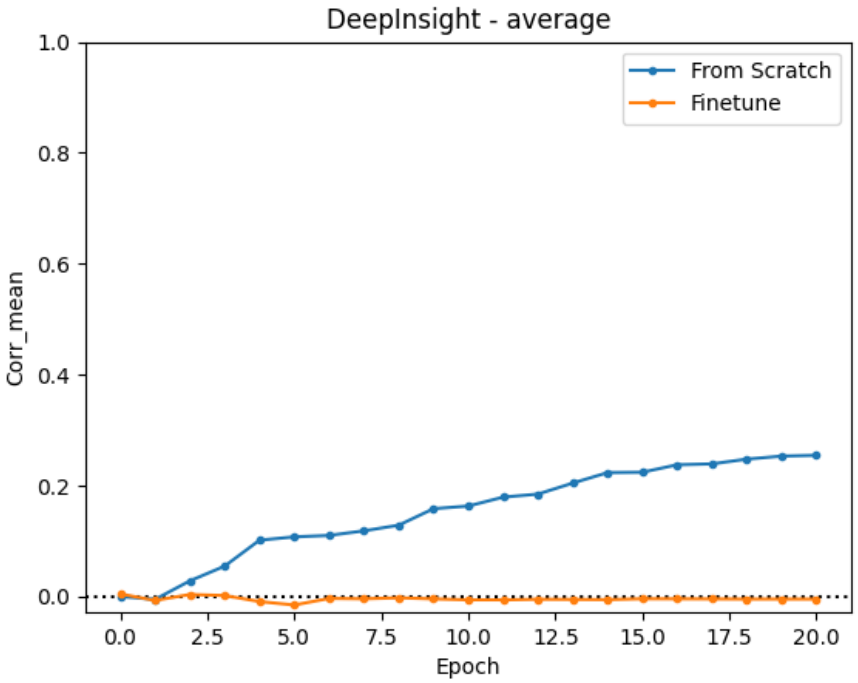


Figure 5.10: Average DeepInsight mean correlation over epochs

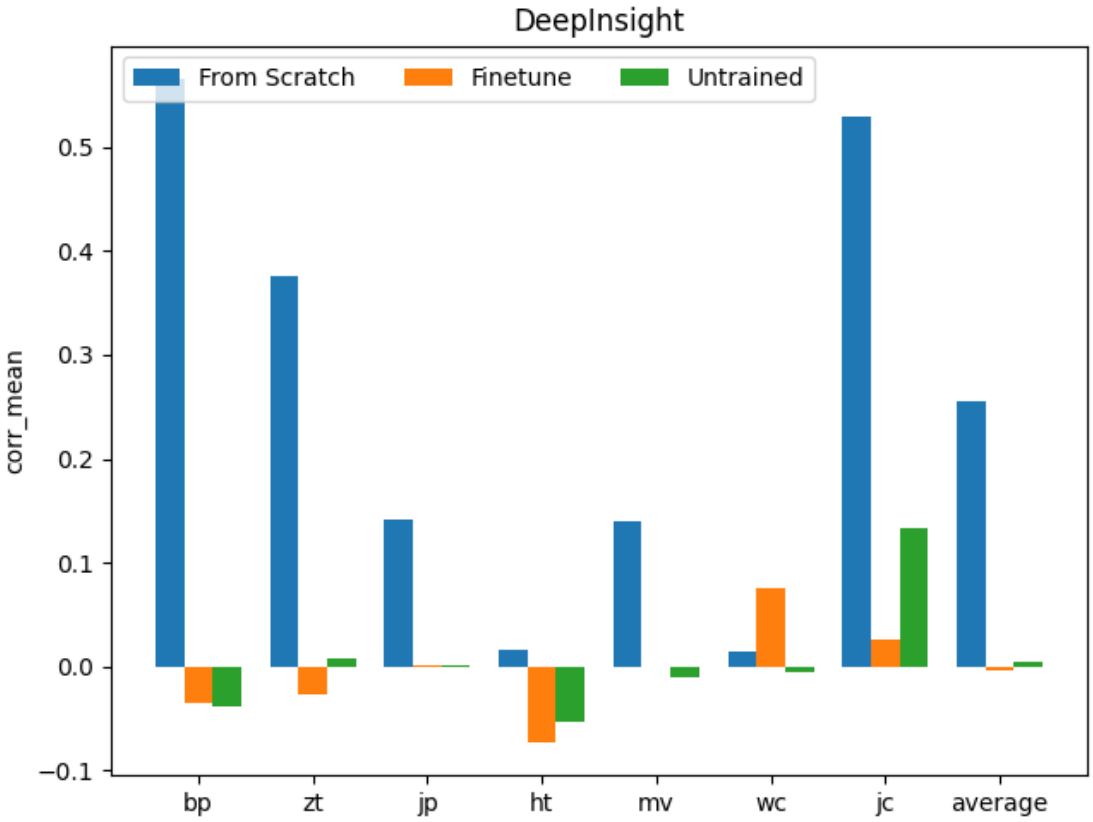


Figure 5.11: DeepInsight performance across subjects

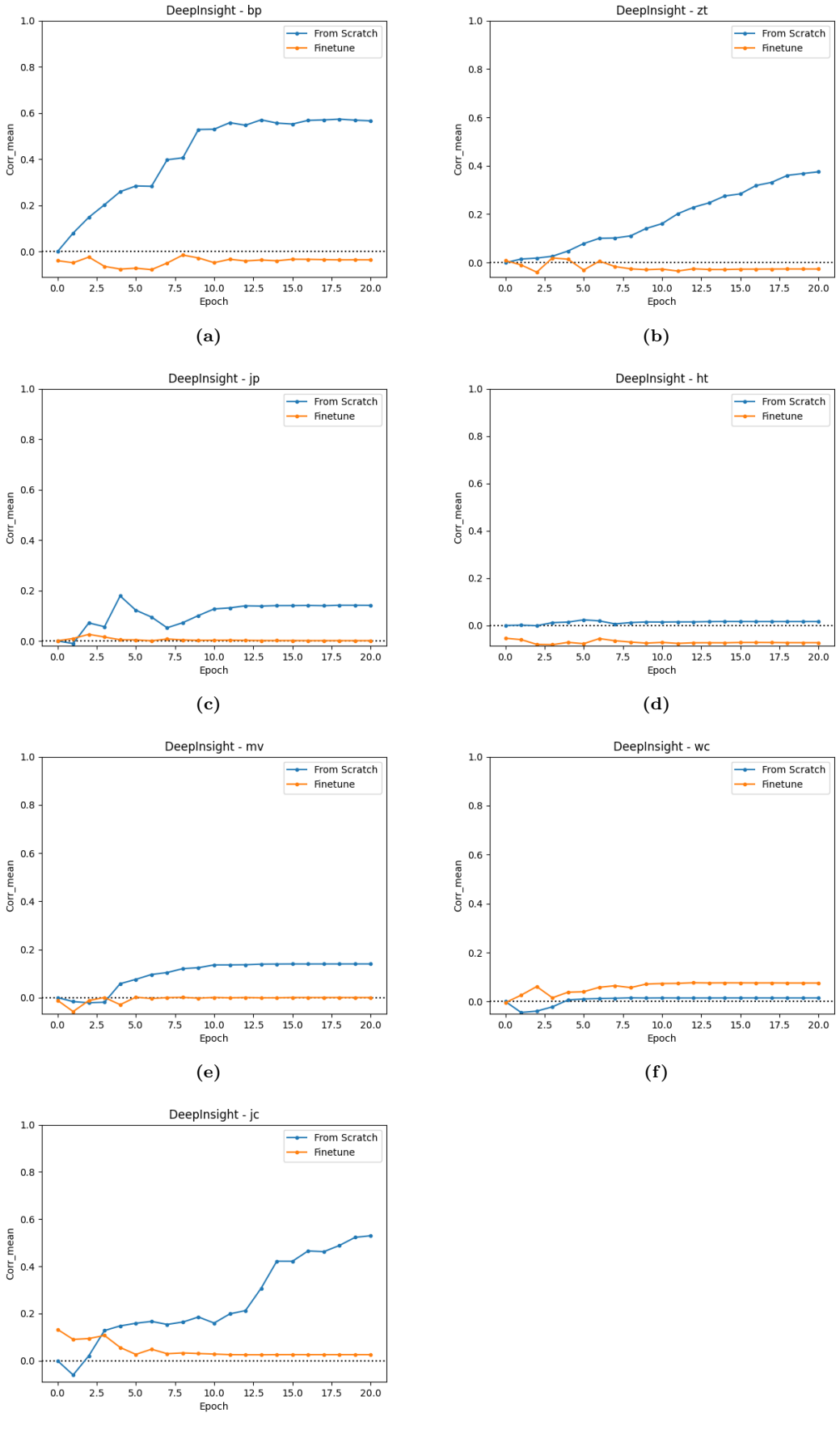


Figure 5.12: DeepInsight mean correlation across subjects over epochs

model	correlation
RandomForest	N/A
SGD	−0.0130
PassiveAggressive	0.2764
Huber	0.2766
SVR	0.3770
AdaBoost	0.4069
HistGradientBoosting	0.4989
HistGradientBoosting*	0.5637

Table 5.4: scikit-learn model mean correlation for subject bp

5.4 HistGradientBoosting

5.4.1 Model Selection

Table 5.4 contains the performance of each regressor tested on subject bp with the preprocessing steps described in section 4.4.2. The random forest regressors test was ended when, after ten hours, the training function had still not returned. Such a long training time would make further testing impractical regardless of model performance.

SGD failed to converge into a functional model.

The PassiveAgressive and Huber regressors perform reasonably well.

SVR and AdaBoost both perform quite well, while HistGradientBoosting has the highest performance of all tested regressors.

Finally, using two HistGradientBoosting models to separately predict movement intensity and which finger moves increases the correlation by 0.07.

5.4.2 Multi-subject Results

The performance of the gradient boosting models is nearly the exact opposite of that of the FingerFlex models.

Figure 5.13 shows a sample of the various models predictions on subject bp. Each prediction was individually rescaled as it varies significantly and would otherwise make the graphs unreadable. We can see that the general model has little correlation to the finger movements, while the finetuned and scratch models both perform reasonably well. The model trained from scratch fits slightly better to the finger movements.

As shown in fig. 5.14 the single-subject models make significant leaps in performance in the first few iterations, while the models previously trained on multiple subjects have a much more gradual rate of improvement until they finally perform slightly worse than the single-subject models. Most graphs in fig. 5.16 follow the same behavior as the average graph in fig. 5.14.

Figure 5.16c is the largest outlier from this pattern, the single-subject model has difficulty early on but eventually outpaces the multi-subject model performance after around 20 iterations. Meanwhile the multi-subject model has relatively high performance for the subject without finetuning, and remains relatively constant throughout the finetuning process. It seems likely the multi-subject model identified a pattern that applies to subject jp but is difficult to discover on only subject jp.

In figs. 5.16d and 5.16g the general model has a slight negative correlation which is corrected over time but does slow down the overall growth of the correlation.

Figure 5.15 and table 5.5 show the performance of each multi-subject model on the new subject before and after finetuning.

Figure 5.13: GradientBoosting predictions on subject bp

Patient Code	From Scratch	Finetune	Untrained
bp	0.5625	0.5598	0.0850
zt	0.3580	0.3321	−0.0171
jp	0.1336	0.1090	0.0690
ht	0.2169	0.1791	−0.0928
mv	0.3083	0.3210	0.0254
wc	0.2182	0.2171	−0.0126
jc	0.2299	0.2177	−0.0815
average	0.2896	0.2765	−0.0035

Table 5.5: GradientBoosting mean correlation per subject

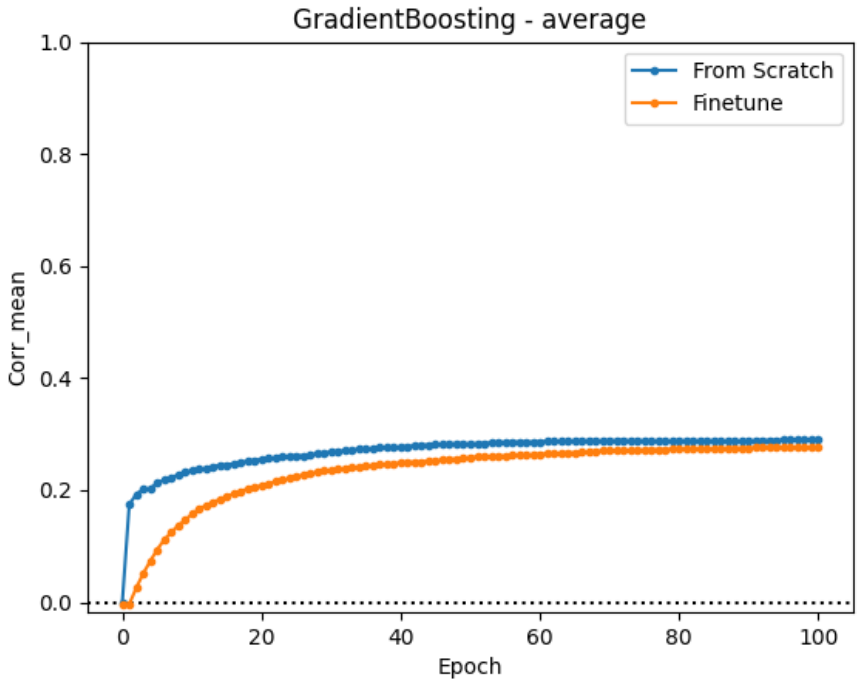
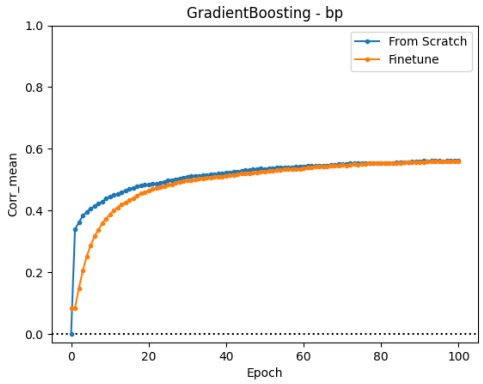
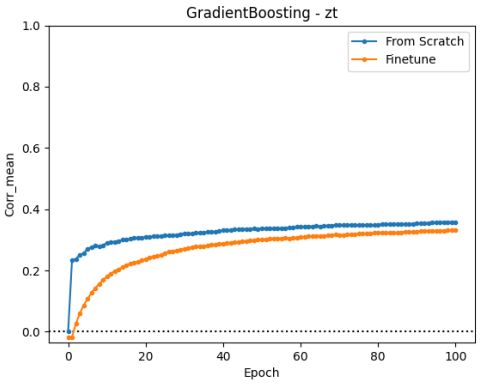


Figure 5.14: Average GradientBoosting mean correlation over epochs

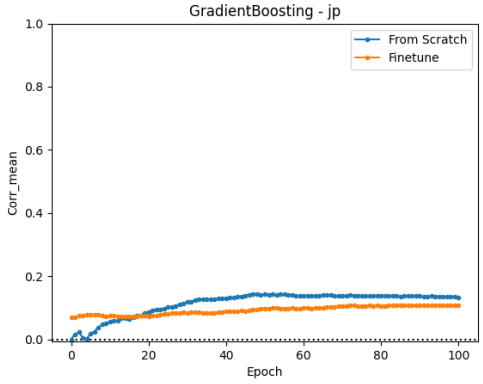
Figure 5.15: GradientBoosting mean correlation across subjects



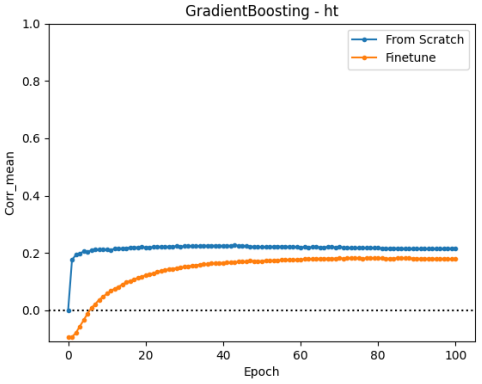
(a)



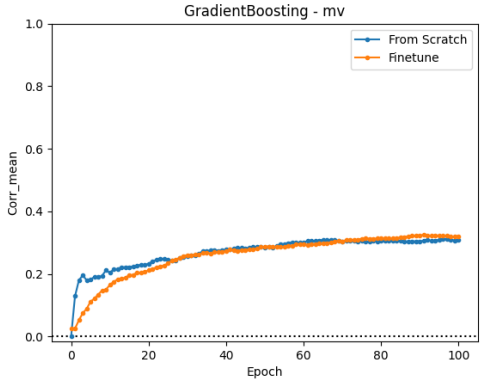
(b)



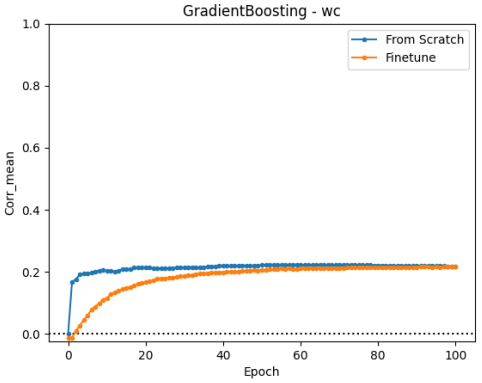
(c)



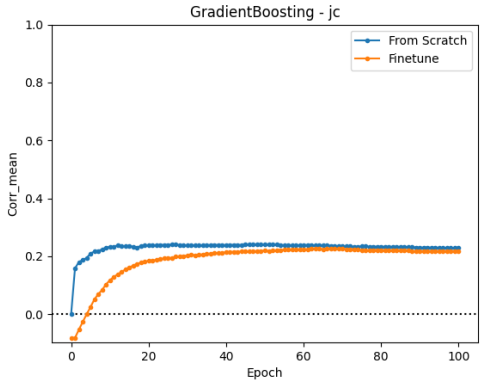
(d)



(e)



(f)



(g)

Figure 5.16: GradientBoosting mean correlation across subjects over epochs

Chapter 6

Standardizing Electrode Positions

Given our belief that the main reason for sub-par performance is the variation in electrode mappings between subjects, we decided to make some initial attempts at standardizing the electrode positions to create a better general model and have had some success.

Our first two attempts tried to directly map positions from one subject to another using the location data available in the Stanford dataset(Miller 2019). First we simply mapped each electrode of the target subject to the closest electrode from the original subject, this resulted in even worse finetune performance. Next we tried calculating the response at the target positions based on the signal measured by the original electrodes. For each target electrode t in the target subject, we calculate $t = \sum_{n=1}^{m} \frac{e_n}{dist(e_n,t)^2}$. With m the total number of origin electrodes, e_n the n'th electrode of the original subject, and $dist(e_n, t)$ a function which calculates the euclidean distance between the two electrodes. This method also resulted in lowered finetune performance.

We tried to train a single dense layer neural network, but found it difficult to provide a working loss function. Since each subject has different finger movements, we cannot simply use the target subject's ECoG or Fingerflex data as a training target. We therefore tried to include a GradientBoosting model trained on the target subject into the loss function for the neural network. This way the neural network would receive the preprocessed ECoG data of a subject as input, and be trained to output preprocessed ECoG data in such a way that the GradientBoosting model can use it to predict finger movements. Unfortunately the library we used did not support such a complicated training pipeline and we decided end the attempt there as it would take to much time to develop for an experiment which is not the main focus of this work.

Finally, we used a channel selection algorithm which optimizes for a pretrained GradientBoosting model. An exhaustive search of all n! permutations of channels is impractical, so we use a greedy algorithm. It iterates over each of the target channels and for each of the original subject's channels tests the model performance and saves the channel which results in the highest correlation. This process is then repeated multiple times, since the choice of 5th channel can alter which channel adds the most correlation as the 1st channel. We chose to only repeat the process thrice as this already takes significantly more time than the actual training of the models. Interestingly, this caused little performance improvement when used to standardize the multi-subject data, but when used to map single-subject data to the multi-subject model after it was trained the resulting correlation is only slightly less than that of the single-subject model.

Table 6.1 and figs. 6.1 and 6.2 show the results of this method of remapping the subject electrode

Patient Code	From Scratch	Finetune	Untrained
bp	0.5678	0.5474	0.3082
zt	0.3528	0.3558	0.1678
jp	0.1517	0.0031	0.0870
ht	0.2208	0.2718	0.2135
mv	0.3083	0.3283	0.0997
wc	0.2284	0.2052	0.1050
jc	0.2336	0.1995	0.1215
average	0.2948	0.2730	0.1575

Table 6.1: GradientBoosting mean correlation per subject, channels remapped

channels. Aside from the anomaly with subject jp all subjects follow the same progression: high initial correlation followed by gradual improvement during finetuning. It should be noted that since it takes significant computing time to map the channels, this method is still worse than just training from scratch. But the results of this experiment imply a general model is practical with better electrode standardization and that the current mismatch indeed severely impacts the multi-subject performance of the examined models.

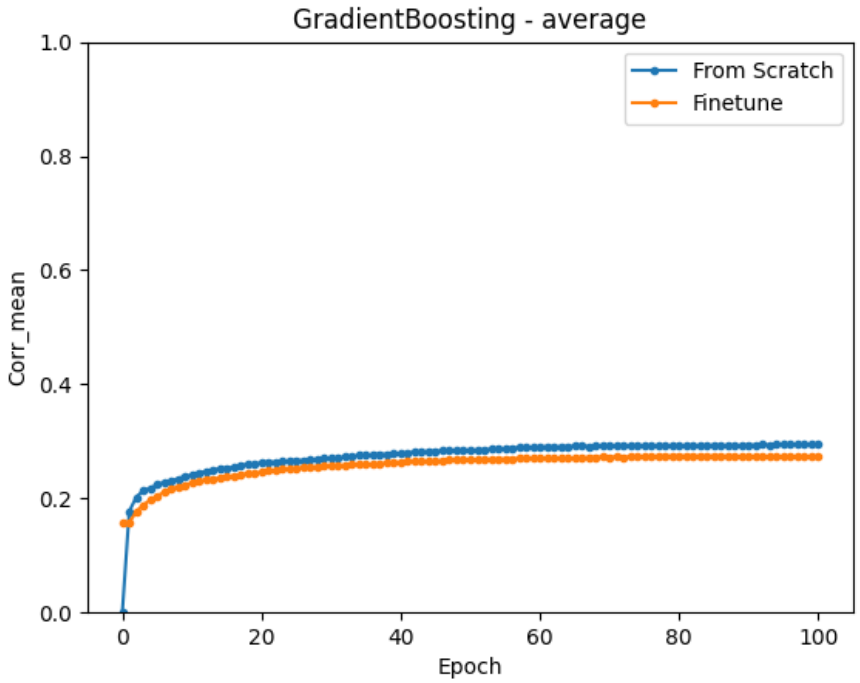


Figure 6.1: Average GradientBoosting mean correlation over epochs, channels remapped

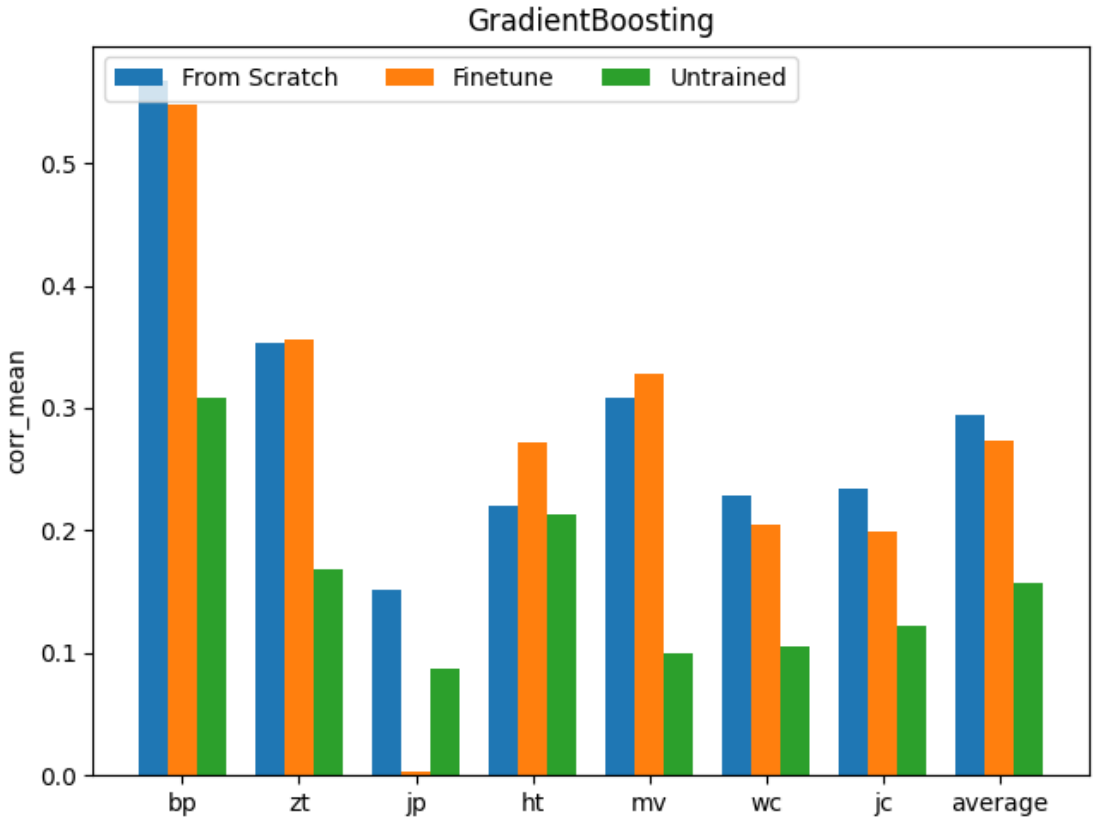
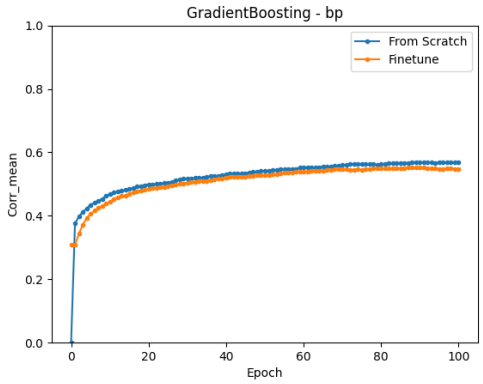
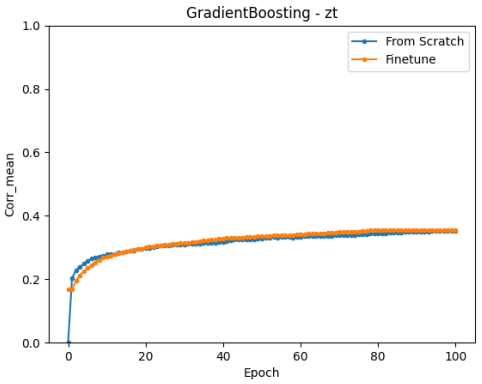


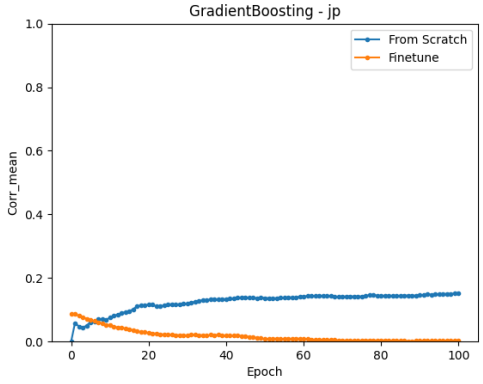
Figure 6.2: GradientBoosting mean correlation across subjects, channels remapped



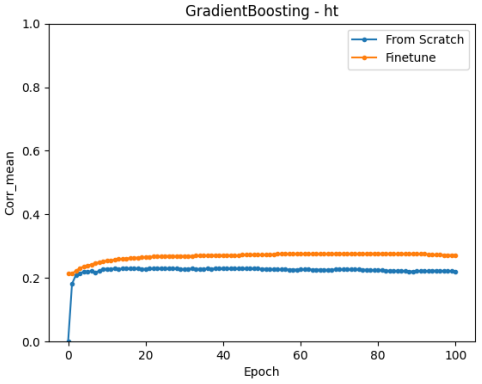
(a)



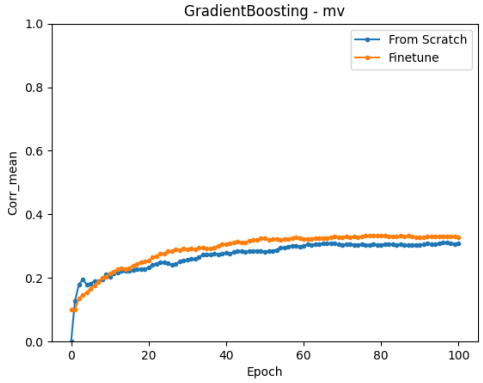
(b)



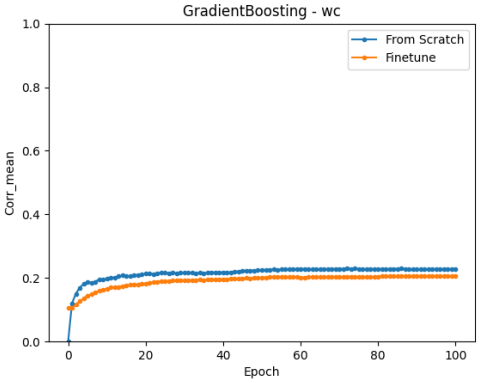
(c)



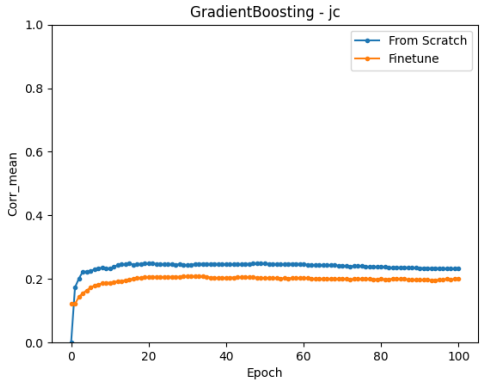
(d)



(e)

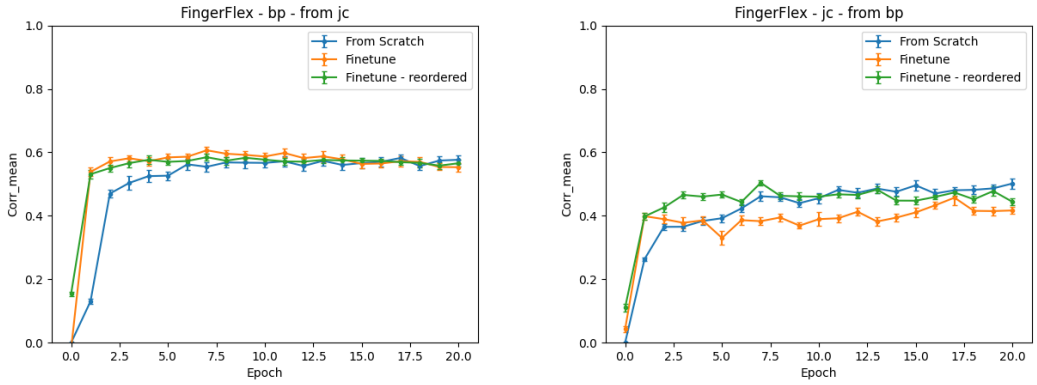


(f)



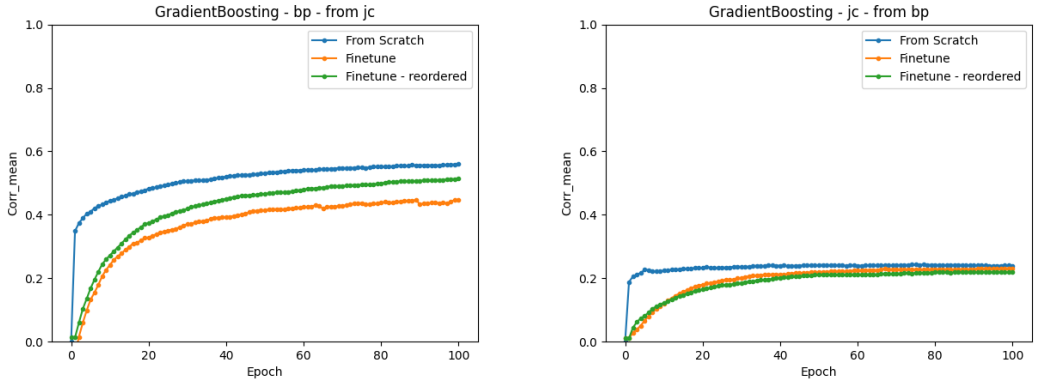
(g)

Figure 6.3: GradientBoosting mean correlation across subjects over epochs, channels remapped



(a) **(b)**

Figure 6.4: Fingerflex mean correlation across subjects over epochs, jc channels remapped



(a) **(b)**

Figure 6.5: GradientBoosting mean correlation across subjects over epochs, jc channels remapped

Back in section 2.3 we also noted the similarity in electrode positions of subject bp and subject jc. We decided to test these two subjects, reversing the order of subject jc's electrodes and measure if this causes any notable improvement. We used the same experimental setup as discussed in chapter 4, only this time with only two subjects meaning a model is trained on a single subject and then further finetuned on the other. We chose to only test this remapping on two methods due to time constraints. Fingerflex was chosen due to it's high adaptability and gradient boosting due to the opposite, requiring high similarity between subjects.

Across all tests we can see that reordering the channels in subject jc increases the performance of the general model before finetuning, and slightly improves overall performance. Though as shown in figs. 6.4a and 6.5b this performance improvement is minimal or nonexistent in some cases. Unfortunately fig. 6.5 still shows general model and finetune performance worse than training a model from scratch, which was not an issue in figs. 6.1 to 6.3 where we remapped the channels for optimal performance. We believe that the electrode grids, while similarly shaped and ordered, still vary in location of the underlying brain regions, and that further refinements would need to identify these locations in addition to purely positional remapping.

Chapter 7

Conclusions

An extensive dataset is important in all machine learning applications and BCI has the difficulty of requiring invasive procedures in order to collect high quality data. Being able to train a machine learning model on data from multiple different people greatly increases the amount of data it can work with and learn from. In this work we've evaluated a variety of state-of-the-art techniques for predicting finger movements from ECoG recordings. This particular task has datasets with a dozen subjects and only ten minutes of recordings per subject. It was therefore an ideal scenario to evaluate how well these models generalize.

In order to evaluate these state-of-the-art models we trained them on multi-subject datasets and evaluate their performance on a single subject excluded from the training dataset. The model's performance is then compared to that of the same model trained on only a single subject, and the model is finetuned for an equal amount of time as the single-subject model is trained.

For an evaluation with minimal adaptations to the techniques themselves, the results we've achieved are highly promising for at least one technique. As shown in section 5.1 the FingerFlex model responds well to multi-subject training with finetuning. Multi-subject training has shown to result in increased performance and significantly shortened training times on new subjects. Our results imply that a significant portion of what a model learns can be generalized and applied to new unseen subjects. A truly general model using this method is currently not yet viable, as performance on unseen subjects without any finetuning is highly variable and generally low. With further research, which we discuss in chapters 6 and 8, these results may be improved for FingerFlex and achieved in other models.

DeepInsight, despite a general similarity to the model architecture Fingerflex uses achieved extremely disappointing results, with it failing to produce a prediction with any correlation at all to the true finger movements. But these findings can help narrow down what makes Fingerflex perform where DeepInsight does not.

BTTR and GradientBoosting approaches are not easily retrained and our experiments reflect that, as these models struggle to adapt to the change in electrode locations across subjects, though we believe that with further work these positions can be better standardized and generalization performance and overall performance will increase.

We conclude that neural network based machine learning techniques show immediate promise in multi-subject training, with the main advantages being the lowered training time required finetuning a pretrained model to a new subject compared to training a model from scratch and a slightly increased overall model performance.

Chapter 8

Future Work

Throughout our work on this project we have encountered many opportunities for further research that had to be left unexplored. Since the main purpose of this thesis was to evaluate how well current state-of-the-art models can make use of data sourced from other subjects, any modifications to the models have been kept to the minimum needed to be able to collect the needed measurements.

The possibility that stood out the most was investigating a method to standardize the ECoG data channels better. Since the Stanford dataset includes the location of the electrodes, it may be possible to project the varied electrode channels of subjects onto a more uniform virtual grid such that each subject has the same amount of virtual electrodes in the same locations. Alternatively a small neural network could be added before the existing models which could be used to translate the different electrode layout into that which the model is trained on.

We would also like to test the effects of adding a single dense layer as the first layer for the DeepInsight(Frey et al. 2021) model, to examine if this would bring its performance more in line with that of Fingerflex(Lomtev, Kovalev, and Timchenko 2022)

With the promising results of the FingerFlex model there is also certainly further refining to be done by modifying the model, tuning hyperparameters, and experimenting with different finetuning methods to possibly improve the state-of-the-art.

Additionally, in this work models are pretrained on all other subjects, but the addition of more subjects likely has diminishing returns. Due to the large amount of time and computing power needed to train these models we have been unable to experiment with the amount of other subjects to train on, but initial tests on FingerFlex models trained on only subject cc show comparative performance to the multi-subject dataset used in this work. Figure 8.1 shows the performance of a model trained on only subject cc, which is closer to that of the single-subject model than the multi-subject model, yet retains the significantly lower training time.

There are also many more models we did not have the time to evaluate which might gain from this technique.

Finally, we are curious about the possibility of training a model on high detail data and then finetuning or otherwise modifying it such that the patterns learned can be applied to lower detail data, such as a subject with less implanted electrodes. HTNet (Peterson et al. 2020) has shown the possibility of training on ECoG data generalizing to EEG data. Since EEG is less invasive than ECoG, expanding on this would allow for these models to require less invasive recording methods when used by patients. It also opens up the possibility of expanding on the current datasets with EEG data, which is easier to collect as subject would not need implants.

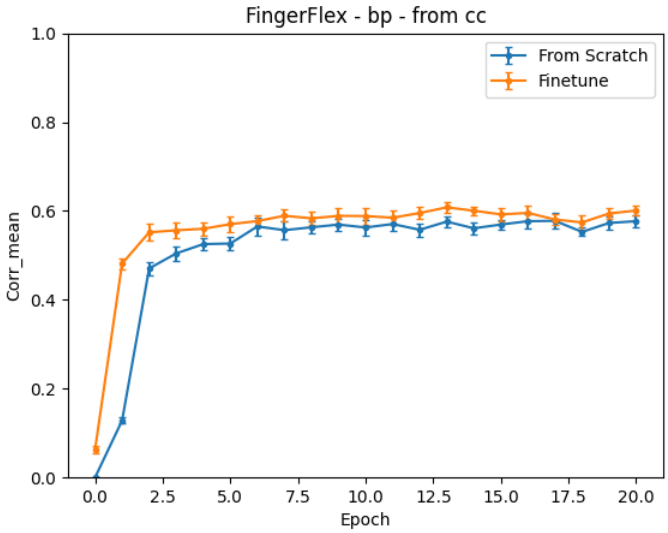


Figure 8.1: FingerFlex mean correlation for subject bp of a model trained on subject cc, over epochs

Bibliography

Xie, Ziqian, Odelia Schwartz, and Abhishek Prasad (Feb. 2018). "Decoding of finger trajectory from ECoG using deep learning". In: *Journal of Neural Engineering* 15.3, p. 036009. ISSN: 1741-2552. DOI: 10.1088/1741-2552/aa9dbe. URL: http://dx.doi.org/10.1088/1741-2552/aa9dbe.

Pan, Jiahui et al. (Dec. 2022). "Advances in P300 brain–computer interface spellers: toward paradigm design and performance evaluation". In: *Frontiers in Human Neuroscience* 16. ISSN: 1662-5161. DOI: 10.3389/fnhum.2022.1077717. URL: http://dx.doi.org/10.3389/fnhum.2022.1077717.

Willett, Francis R. et al. (May 2021). "High-performance brain-to-text communication via handwriting". In: *Nature* 593.7858, pp. 249–254. ISSN: 1476-4687. DOI: 10.1038/s41586-021-03506-2. URL: http://dx.doi.org/10.1038/s41586-021-03506-2.

Metzger, Sean L. et al. (Aug. 2023). "A high-performance neuroprosthesis for speech decoding and avatar control". In: *Nature* 620.7976, pp. 1037–1046. ISSN: 1476-4687. DOI: 10.1038/s41586-023-06443-4. URL: http://dx.doi.org/10.1038/s41586-023-06443-4.

Hochberg, Leigh R. et al. (July 2006). "Neuronal ensemble control of prosthetic devices by a human with tetraplegia". In: *Nature* 442.7099, pp. 164–171. ISSN: 1476-4687. DOI: 10.1038/nature04970. URL: http://dx.doi.org/10.1038/nature04970.

Vaid, Swati, Preeti Singh, and Chamandeep Kaur (Feb. 2015). "EEG Signal Analysis for BCI Interface: A Review". In: *2015 Fifth International Conference on Advanced Computing amp; Communication Technologies*. IEEE, pp. 143–147. DOI: 10.1109/acct.2015.72. URL: http://dx.doi.org/10.1109/ACCT.2015.72.

Schalk, G et al. (June 2007). "Decoding two-dimensional movement trajectories using electrocorticographic signals in humans". In: *Journal of Neural Engineering* 4.3, pp. 264–275. ISSN: 1741-2552. DOI: 10.1088/1741-2560/4/3/012. URL: http://dx.doi.org/10.1088/1741-2560/4/3/012.

Blausen.com staff (2014). "Medical gallery of Blausen Medical 2014". In: *WikiJournal of Medicine* 1.2. ISSN: 2002-4436. DOI: 10.15347/wjm/2014.010. URL: http://dx.doi.org/10.15347/wjm/2014.010.

Hashiguchi, Kimiaki et al. (Apr. 2007). "Correlation between scalp-recorded electroencephalographic and electrocorticographic activities during ictal period". In: *Seizure* 16.3, pp. 238–247. ISSN: 1059-1311. DOI: 10.1016/j.seizure.2006.12.010. URL: http://dx.doi.org/10.1016/j.seizure.2006.12.010.

Sugano, Hidenori, Hiroyuki Shimizu, and Shigeki Sunaga (Mar. 2007). "Efficacy of intraoperative electrocorticography for assessing seizure outcomes in intractable epilepsy patients with temporal-lobe-mass lesions". In: *Seizure* 16.2, pp. 120–127. ISSN: 1059-1311. DOI: 10.1016/j.seizure.2006.10.010. URL: http://dx.doi.org/10.1016/j.seizure.2006.10.010.

Miller, Kai J. (Aug. 2019). "A library of human electrocorticographic data and analyses". In: *Nature Human Behaviour* 3.11, pp. 1225–1235. ISSN: 2397-3374. DOI: 10.1038/s41562-019-0678-3. URL: http://dx.doi.org/10.1038/s41562-019-0678-3.

Miller, Kai J. et al. (Sept. 2012). "Human Motor Cortical Activity Is Selectively Phase-Entrained on Underlying Rhythms". In: *PLoS Computational Biology* 8.9. Ed. by Tim Behrens, e1002655. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1002655. URL: http://dx.doi.org/10.1371/journal.pcbi.1002655.

Flamary, Rémi and Alain Rakotomamonjy (2012). "Decoding Finger Movements from ECoG Signals Using Switching Linear Models". In: *Frontiers in Neuroscience* 6. ISSN: 1662-4548. DOI: 10.3389/fnins.2012.00029. URL: http://dx.doi.org/10.3389/fnins.2012.00029.

Liang, Nanying and Laurent Bougrain (2012). "Decoding Finger Flexion from Band-Specific ECoG Signals in Humans". In: *Frontiers in Neuroscience* 6. ISSN: 1662-4548. DOI: 10.3389/fnins.2012.00091. URL: http://dx.doi.org/10.3389/fnins.2012.00091.

Sanchez, Justin C et al. (2008). "Extraction and localization of mesoscopic motor control signals for human ECoG neuroprosthetics". In: *Journal of neuroscience methods* 167.1, pp. 63–81.

Frey, Markus et al. (Aug. 2021). "Interpreting wide-band neural activity using convolutional neural networks". In: *eLife* 10. ISSN: 2050-084X. DOI: 10.7554/elife.66551. URL: http://dx.doi.org/10.7554/eLife.66551.

Yao, Lin, Bingzhao Zhu, and Mahsa Shoaran (Feb. 2022). "Fast and accurate decoding of finger movements from ECoG through Riemannian features and modern machine learning techniques". In: *Journal of Neural Engineering* 19.1, p. 016037. ISSN: 1741-2552. DOI: 10.1088/1741-2552/ac4ed1. URL: http://dx.doi.org/10.1088/1741-2552/ac4ed1.

Faes, Axel, Flavio Camarrone, and Marc M. Van Hulle (July 2024). "Single Finger Trajectory Prediction From Intracranial Brain Activity Using Block-Term Tensor Regression With Fast and Automatic Component Extraction". In: *IEEE Transactions on Neural Networks and Learning Systems* 35.7, pp. 8897–8908. ISSN: 2162-2388. DOI: 10.1109/tnnls.2022.3216589. URL: http://dx.doi.org/10.1109/TNNLS.2022.3216589.

Ludwig, Kip A. et al. (Mar. 2009). "Using a Common Average Reference to Improve Cortical Neuron Recordings From Microelectrode Arrays". In: *Journal of Neurophysiology* 101.3, pp. 1679–1689. ISSN: 1522-1598. DOI: 10.1152/jn.90989.2008. URL: http://dx.doi.org/10.1152/jn.90989.2008.

Lomtev, Vladislav, Alexander Kovalev, and Alexey Timchenko (2022). *FingerFlex: Inferring Finger Trajectories from ECoG signals*. DOI: 10.48550/ARXIV.2211.01960. URL: https://arxiv.org/abs/2211.01960.

Schneider, Steffen et al. (Sept. 2019). "wav2vec: Unsupervised Pre-Training for Speech Recognition". In: *Interspeech 2019*. interspeech$_2$019. ISCA. DOI: 10.21437/interspeech.2019-1873. URL: http://dx.doi.org/10.21437/Interspeech.2019-1873.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation*. DOI: 10.48550/ARXIV.1505.04597. URL: https://arxiv.org/abs/1505.04597.

De Lathauwer, Lieven and Dimitri Nion (Jan. 2008). "Decompositions of a Higher-Order Tensor in Block Terms—Part III: Alternating Least Squares Algorithms". In: *SIAM Journal on Matrix Analysis and Applications* 30.3, pp. 1067–1083. ISSN: 1095-7162. DOI: 10.1137/070690730. URL: http://dx.doi.org/10.1137/070690730.

Yokota, Tatsuya and Andrzej Cichocki (2014). "Multilinear tensor rank estimation via sparse Tucker decomposition". In: *2014 Joint 7th International Conference on Soft Computing and Intelligent Systems (SCIS) and 15th International Symposium on Advanced Intelligent Systems (ISIS)*. IEEE, pp. 478–483.

De Lathauwer, Lieven, Bart De Moor, and Joos Vandewalle (2000a). "A multilinear singular value decomposition". In: *SIAM journal on Matrix Analysis and Applications* 21.4, pp. 1253–1278.

Zubair, Syed and Wenwu Wang (2013). "Tensor dictionary learning with sparse tucker decomposition". In: *2013 18th international conference on digital signal processing (DSP)*. IEEE, pp. 1–6.

Phan, Anh Huy and Andrzej Cichocki (2008). "Fast and efficient algorithms for nonnegative Tucker decomposition". In: *International Symposium on Neural Networks*. Springer, pp. 772–782.

De Lathauwer, Lieven, Bart De Moor, and Joos Vandewalle (2000b). "On the best rank-1 and rank-(r 1, r 2,..., rn) approximation of higher-order tensors". In: *SIAM journal on Matrix Analysis and Applications* 21.4, pp. 1324–1342.

Allen, Genevera I and Mirjana Maletić-Savatić (2011). "Sparse non-negative generalized PCA with applications to metabolomics". In: *Bioinformatics* 27.21, pp. 3029–3035.

Ke, Guolin et al. (2017). "LightGBM: A Highly Efficient Gradient Boosting Decision Tree". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

Peterson, Steven M. et al. (Nov. 2020). "Generalized neural decoders for transfer learning across participants and recording modalities". In: DOI: 10.1101/2020.10.30.362558. URL: http://dx.doi.org/10.1101/2020.10.30.362558.