Few-Shot Classification of EEG with Quasi-Inductive Transfer Learning

Mathias Duedahl

Bachelor's thesis
2022:K22

Lund University

Faculty of Science
Centre for Mathematical Sciences
Mathematical Statistics

CENTRUM SCIENTIARUM MATHEMATICARUM

Bachelor’s Theses in Mathematical Sciences 2022:K22
ISSN 1654-6229

LUNFMS-4068-2022

Mathematical Statistics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lu.se/

Supervised by
Prof. Maria SANDSTEN
Ph.D. Student Oskar KEDING

Acknowledgements

I would like to thank my supervisor, Maria Sandsten, and my co-supervisor, Oskar Keding, for their guidance and support throughout this project. Their advice and feedback were very valuable, and I appreciate their expertise and mentorship.

I would also like to thank Mikael Johansson for sharing the data used in this project.

Lastly, I would like to thank my place of work for providing me with the necessary resources to complete this project.

I am grateful for the contributions of these individuals, and their support has been crucial to the success of this project.

Abstract

Brain-computer interfaces (BCIs) are devices that enable people with disabilities to use their thoughts to control external devices and restore or improve their bodily functions. One important aspect of BCIs is the classification of electroencephalography (EEG) signals, which measure brain activity and can be difficult to interpret. To address this challenge, we use time-frequency transformations to convert EEG signals into images and employ pre-trained deep convolutional neural networks (CNNs) to classify the images based on whether the subject heard a sound from the left or the right ear. We investigate whether transfer learning, a technique that involves using a pre-trained model on a related task as the starting point for training on a new task, is effective even when the source and target domains are very different. The best classification result achieved was 61.7%, using EfficientNet V2 tuned on 5 different test-subjects, and tuned on the target subject.

Contents

Chapter 1

Introduction

Brain-Computer Interface (BCI), is a field of research aiming to develop technologies that allow for direct communication between the brain and external devices, a key application of BCI is aiding people with disabilities by enhancing their ability to interact with their environment. Electroencephalography (EEG) is a widely used technique in neuroscience for recording the electrical activity of the brain, and is a commonly used source of data in BCI technology [27].

In order to use EEG data to control external devices, it must first be analyzed to determine the intended action. Time-Frequency analysis is a powerful tool for studying the dynamics of the brain activity, as it allows for the separation of the EEG signal into different frequency bands [21]. However, the analysis of Time-Frequency transformed EEG data can be challenging, as it is no trivial matter to learn patterns in this kind of data, so we take advantage of the tools at our disposal - we let the computer do the learning.

Transfer learning is a powerful tool in machine learning that allows a model trained on one task to be fine-tuned and reused on a different but related task. In this thesis, we explore the application of transfer learning to classify Time-Frequency transformed EEG data using pre-trained deep convolutional neural networks.

In this thesis, we set out to classify Time-Frequency transformed EEG data from subjects receiving auditory stimuli, and determine from which side they heard the sound. To do this, we employ inductive (different task) transfer learning, by using deep convolutional neural networks that have been pre-trained on ImageNet, the target task is different, in that we are performing a binary classification, whereas the ImageNet dataset has hundreds of different classifications.

More notably, the source domain from which the classifier input data originates is quite different, since ImageNet consists of photos from the real world. The only real similarity between the Time-Frequency transformed EEG data, and ImageNet, is that they are both images and perhaps share generic features present in natural images. This inherent difference between the source and target domains, is non-standard in this type of classification task, and will be one of the main challenges we hope to overcome in this thesis.

This constitutes an additional point of interest, will this non-standard, or quasi-inductive, transfer learning yield useful results.

All the code used throughout this thesis, to train and evaluate models, visualize results, etc. is accessible on Github:
https://github.com/Lussebullen/EEG_Classification/tree/InterSubTL.

Chapter 2

Theory

In this chapter, we give a high-level description of the machine learning concepts and tools used in this project, we cover the most important aspects of convolutional neural networks, and introduce key few-shot learning techniques. To do that, we first introduce the learning problem, from computational learning theory.

2.1 The Learning Problem

To formally introduce the learning problem, we must first introduce some notation. Let $\mathcal{X}$ denote the set of all possible examples, also called the *feature space*, and let $\mathcal{Y}$ denote the set of all possible labels, i.e. the *label space.* A *concept*, $c : \mathcal{X} \mapsto \mathcal{Y}$, is a mapping from $\mathcal{X}$ to $\mathcal{Y}$, a *concept class* is a set of concepts we have an interest in learning, and is denoted $\mathcal{C}$.

Let us clarify this notation by example, imagine you are an ice cream maker, every ice cream consists of a base (cone or cup), one scoop (any flavor) and a topping (like sprinkles), and you are interested in determining whether an ice cream tastes good or bad. In this learning example, the feature space is the set of all possible ice creams you can construct. The label space would simply be $\{bad, good\}$, the concept class would contain multiple concepts for determining whether an ice cream is good or bad, and a specific concept of interest, $c \in \mathcal{C}$, might be ”does the ice cream taste good or bad to my customers”.

For the formulation of the learning problem, we first assume that examples, $x \in \mathcal{X}$, are independent and identically distributed (i.i.d.) according to an unknown probability distribution $P(x)$.

The learning problem, as formulated by Mohri et al. [20], is then:

> The learner considers a fixed set of possible concepts $\mathcal{H}$, called a hypothesis set, which might not necessarily coincide with $\mathcal{C}$. It receives a sample $X = (x_1, \dots, x_n)$, containing n examples drawn i.i.d. according to $P(x)$ as well as the labels $Y = (c(x_1), \dots, c(x_n)) = (y_1, \dots, y_n)$, which are based on a specific target concept $c \in \mathcal{C}$ to learn. The task is then to use the labeled sample X to select a hypothesis $h_X \in \mathcal{H}$ that has a small generalization error with respect to the concept c.

We will detail this further in the section on backpropagation.

2.2 Convolutional Neural Networks

Multilayer feedforward networks and, its subclass, convolutional neural networks (CNNs), are universal approximators [17].

This means there exists model parameters, such that they are able to approximate any continuous function with arbitrary accuracy, provided the network architecture is chosen appropriately [34].

In this paper the primary focus will be on CNNs. As it turns out, this network variant works extremely well, not just in theory, but in a wide fan of practical applications, particularly within image recognition.

To understand the advantages and drawbacks of CNNs, we first review a basic artificial neural network, namely the multilayer perceptron (MLP), and some of its key mechanisms.

2.2.1 Multilayer Perceptron

A multilayer perceptron consists of an input layer, multiple hidden layers, and an output layer. See figure 2.1 for the general structure of an MLP.

Here, node j in the layer, l, has the weight (w_{jk}^l) associated with the output from node k from layer $(l-1)$, each weight represented by a vertex. The linear combination of weights and the previous layers outputs (activations), along with the associated bias, is then fed into the activation function (f^l), which yields the activation of node j in layer l

$$a_j^l = f^l \left(b_j^l + \sum_k w_{jk}^l a_k^{l-1} \right).$$

This can be simplified with the use of linear algebra, by prepending 1 to each activation-vector, $A^l = (a^l_j)$, and adding a row of weights to account for the bias term

$$A^l = f^l(W^l A^{l-1}), \tag{2.1}$$

where $W^l = (w^l_{jk})$, and f^l is applied elementwise [23].

The possible non-linearity of f^l allows the multilayer perceptron to learn arbitrarily complex decision boundaries [2].

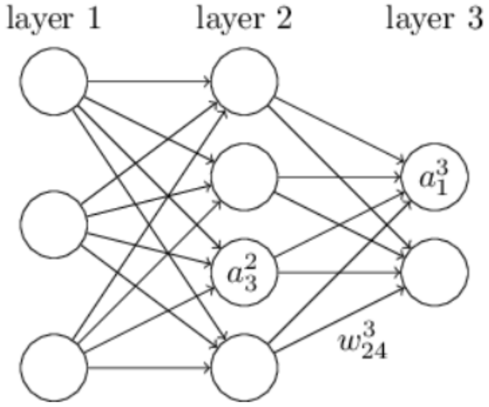


Figure 2.1: A multilayer perceptron displaying how weights and activation notation associated with the nodes and vertices in the network. Layer 1, 2 and 3 are the input layer, a hidden layer, and the output layer respectively.

In an MLP, we can feed our data into the input layer, after which we pass it forward through each layer, and at the output layer we get an interpretable output.
Thus far, we have just created a network of nodes that, when given data, will output probabilities for a desired amount of classes. Unfortunately, these probabilities aren't exactly worth much yet, since the network is initialized with random weights. We have to learn the weights, such that, when given data, the network outputs probabilities that accurately represent the label(s) of the data.

2.2.2 Backpropagation

From the learning problem, we wish to achieve a small generalization error, or *risk*, of a hypothesis $h \in \mathcal{H}$. The risk is dependent on how we quantify the performance of our model.

To quantify the performance of our model, we choose a *loss function*, $\ell(y, \hat{y})$, that given the true value of our example, $y = c(x) \in \mathcal{Y}$ for an example

$x \in X$, and the model prediction $\hat{y} = h(x; W)$, will give us a measure of the model error, or loss.

We want a model that generalizes well, and has low loss everywhere, so we introduce the *empirical risk* function, which is the mean loss for all of our training example/label pairs from our sample, (x_i, y_i) for all $i \in \{1, \ldots, n\}$,

$$J(W) = \frac{1}{n} \sum_{i=1}^{n} \ell\Big(y_i, h(x_i; W)\Big). \tag{2.2}$$

The empirical risk function is also known as the objective function, this is what we wish to minimize.[1]

This is what we use backpropagation for, it is a method of computing the objective function gradient, so that we can adjust the model weights to minimize ℓ.

The core principle of backpropagation is the chain rule, this results from equation 2.1, as this is one forward pass, calculating a layer from the previous layer outputs. Let L and x denote the total amount of layers in the network and an input example respectively, then the network can be represented as

$$h(x; W) = f^L(W^L f^{L-1}(\cdots f^1(W^1 x) \cdots)).$$

This is clearly a composite function, meaning we need to use the chain rule to determine the gradient of the objective function from equation 2.2 [2].

We can use this gradient to nudge the weights towards a local minimum, see figure 2.2, in this example we only have one parameter, as you can imagine with many weights, the objective function landscape will grow very complicated.

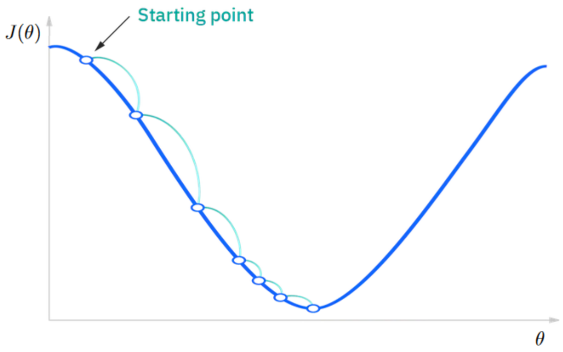


Figure 2.2: Gradient descent with only one parameter θ [10].

[1] Empirical risk is colloquially referred to as loss, loss often appears as an axis label in graphs displaying the empirical risk.

2.2.3 Convolutional Layer

What really sets apart CNNs from the MLP is the convolutional layer, this layer is quite adept at picking out features from images. To see why the convolutional layer is so well suited for detecting features in images, let us consider an example.

A convolutional layer consists of an arbitrary $m \times n$ filter and various other, less pertinent for our purposes, parameters. See figure 2.3, here we have a 3×3 filter that is applied to an input image (represented as a matrix), resulting in the output array, also called the feature map.

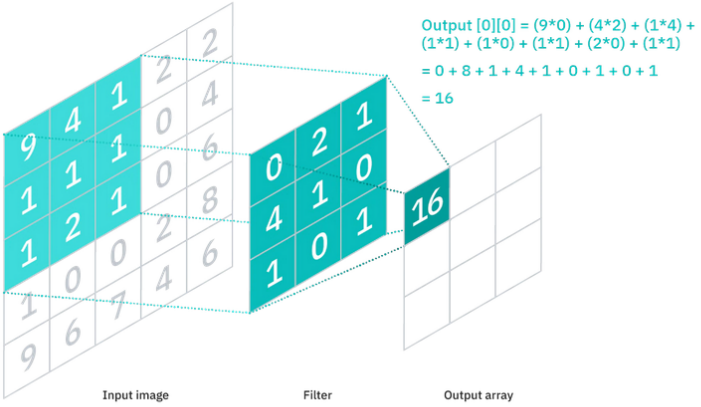

Output [0][0] = (9*0) + (4*2) + (1*4) + (1*1) + (1*0) + (1*1) + (2*0) + (1*1)
= 0 + 8 + 1 + 4 + 1 + 0 + 1 + 0 + 1
= 16
16
Input image
Filter
Output array

Figure 2.3: First calculation in a convolutional layer [9].

The convolution operation here consists of ”sliding” the filter over the input image, and calculating the sum of the elementwise products for each possible position of the filter.

Now consider the example in figure 2.4, from the filter we can imagine that at any pixel where all the eight neighbouring pixels have the same color, will result in zero after the convolution.

Thus, we only get non-zero results when there are differences in color around a pixel, and the resulting value deviates further from zero the larger the differences. This results in all the clearly visible edges of the image being highlighted.

Convolution filters can be constructed to perform various different image operations, such as sharpening, blurring, edge detection etc.

This is what CNNs take advantage of, we feed the network images, and have it train to adjust the filter such that is extracts features that improves the networks classification accuracy. This learning of the filter, is also done through backpropagation, only it can be adjusted to function for convolutions [28].

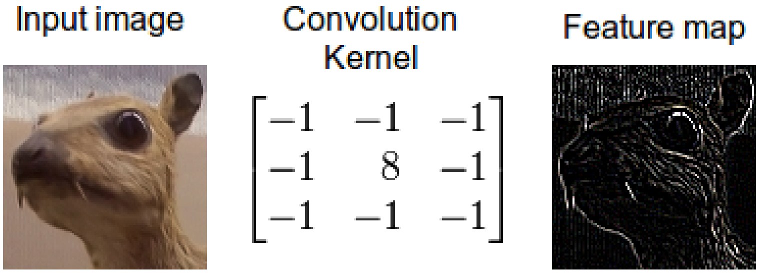

Input image
Convolution Kernel
$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$
Feature map

Figure 2.4: Ridge detection filter applied to image [7].

By adding multiple convolutional layers, it is possible to learn much more advanced features. Filters in the early layers might be able to detect simple features like edges and colours, whereas filters in later layers may be able to detect constructions of features learned in earlier layers, like a certain collection of edges might make up a bicycle [9].

2.3 Few-Shot Learning

Problems arise when you don't have an abundance of data to train your model, one such problem is the vanishing gradient problem.

When training neural networks where backpropagation is used to update the weights, we compute gradients using the chainrule, with a deeper network we end up with a product of many partial derivatives in the initial layers of the model. Many of the most common activation functions have derivative values between 0 and 1, so working our way towards the beginning of a deep network, the gradient risks becoming negligibly small [3].

If we only have access to a small amount of training data, this becomes an issue, as we simply won't be able to train the weights in the early layers of the network, so it becomes unfeasible to train a deep neural network on a small dataset.

This type of learning problem, with little data to learn the intended task, is what we call few-shot learning (FSL) [30].

2.3.1 Overfitting

Given the lack of data, FSL problems are prone to overfitting. Overfitting occurs when the model, instead of learning a pattern that generalizes to never-before-seen data from the same distribution, learns to fit the noise in the training data, see figure 2.5.

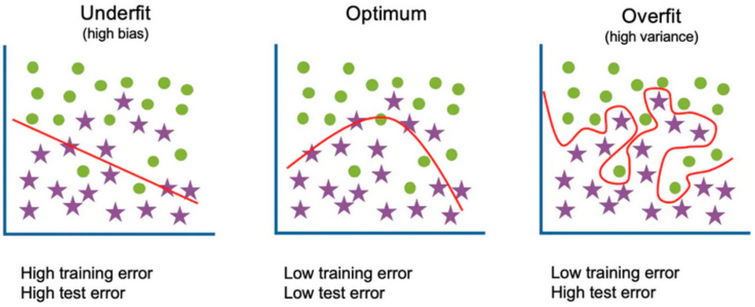


Figure 2.5: Different models' separation boundaries for identical data. In the first (from the left) example, the choice of separation boundary is suboptimal, the second choice fits well to the apparent distribution of the sample, the third choice is fit exactly to the sample, and will likely not perform well on future data from the same data generating distribution [11].

The problem with overfitting is that the model is only good for the training data, where we already know the labels, we want a model that lets us predict accurately on future data.

There are many techniques that aid in the prevention of overfitting, but the most fundamental method is to split your labelled data into three (or two) parts:

1. Training data, used to train model parameters.
2. Validation data (optional), used to train model hyper parameters[2].
3. Test data, used **only** to assess performance of the final model.

This split allows us to monitor the model performance on the training and validation data during model training. If the training error keeps decreasing while the validation error increases, it is a symptom of overfitting, since the model no longer generalizes well, see figure 2.6 [32].

2.3.2 Transfer Learning

As discussed earlier, deep neural networks (DNN), such as CNNs with many layers, are able to learn many features, and can "compound" these into advanced features. The problem with these DNNs is that it takes a lot of data, compute, and time to train them properly. This is simply not possible in the context of FSL.

[2] Parameters that affect the learning process itself, such as the choice of optimizer.

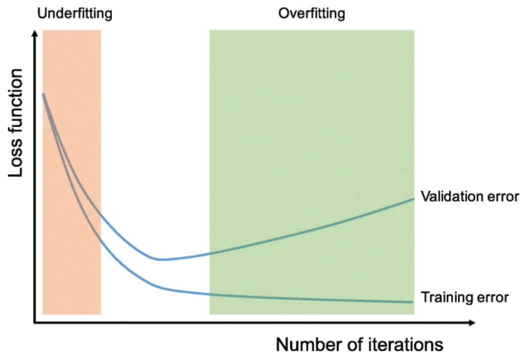


Figure 2.6: Detecting overfitting using training and validation accuracy, when both training and validation error is high, we are underfitting, if the gap between training and validation error becomes too large, we are overfitting, and the model will no longer generalize well [32].

It has become a popular practice to take pre-trained deep neural networks and adjust them to FSL tasks, in order to use the already learned features and simply tune the complex model to fit your needs. This is what we call *transfer learning.*

Definition 2.3.1 (Transfer Learning)**.** A domain $\mathcal{D}$ is comprised of a feature space $\mathcal{X}$ and a marginal probability distribution $P(X)$ where X is a sample. Given a specific domain, $\mathcal{D} = \{\mathcal{X}, P(X)\}$, a task consists of a label space, $\mathcal{Y}$, and an objective predictive function, or concept, $f : \mathcal{X} \mapsto \mathcal{Y}$.

This task, $\mathcal{T} = \{\mathcal{Y}, f(\cdot)\}$, is learned from the training sample example/label pairs.

Given a source domain $\mathcal{D}_S$, learning task $\mathcal{T}_S$, target domain $\mathcal{D}_T$ and learning task $\mathcal{T}_T$, transfer learning aims to help improve the learning of the target predictive function $f(\cdot)$ in $\mathcal{D}_T$ using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$, or $\mathcal{T}_S \neq \mathcal{T}_T$ [33].

There are two widely practiced ways in which to use a pre-trained network, namely fixed feature extraction and fine-tuning, see figure 2.7.

In fixed feature extraction we freeze the weights in the entire model, and introduce a new classifier as the final layer that has been adapted to our task, and then training the model on our data, the training will be localized to only the classifier which reduced the computation time from backpropagation significantly.

In fine-tuning we also replace the final model layer (assuming the pre-trained and desired model are predicting a different amount of classes), but here we do not freeze the entirety of the model, we choose which layers we

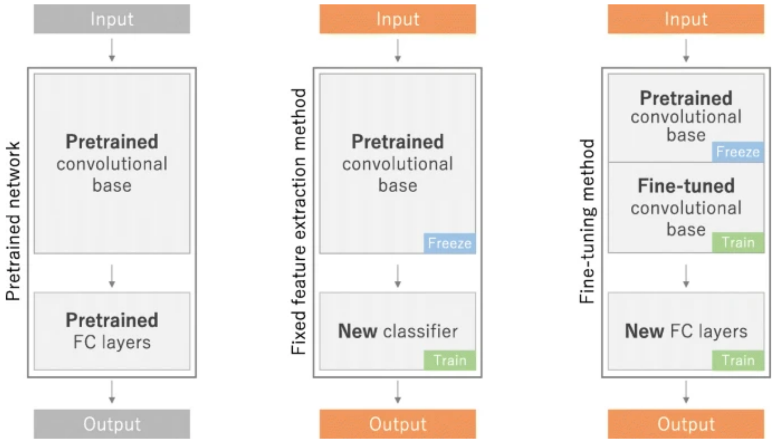

Input
Pretrained network
Pretrained convolutional base
Pretrained FC layers
Output
Input
Fixed feature extraction method
Pretrained convolutional base
Freeze
New classifier
Train
Output
Input
Fine-tuning method
Pretrained convolutional base
Freeze
Fine-tuned convolutional base
Train
New FC layers
Train
Output

Figure 2.7: Pre-trained deep CNN (gray), and two types of transfer learning (coloured), fixed feature extraction and fine-tuning, from the left respectively [32].

want to train on our data, in order to tune the weights in a larger portion of the model to fit our learning problem.

For our purposes, in both the above cases, we adapt the final layer to our target classes. This is the case from definition 2.3.1 where $\mathcal{T}_S \neq \mathcal{T}_T$, and is called *inductive transfer learning* [24].

2.3.3 Data Augmentation

The most immediate problem with FSL is the lack of data, so what better way to solve the problem than to acquire more of it? Unfortunately, this can be a prohibitively difficult endeavour. Luckily, we do have a clever trick to achieve a similar effect to having more data, namely data augmentation.

Data augmentation is simply modifying the data you already have, so that it is different, but still has the desired learnable features. For example by flipping, rotating or adding noise to an image, a lion is still a lion if it is rotated.

Adding augmented data in your training is an effective measure to prevent overfitting, as it prevents the model from seeing the exact same data during each training iteration, and thus stops it from overfitting to certain features specific to the original image [32].

Chapter 3

Method

In this chapter, we examine the data we will be working with, introduce the deep neural networks we will be training. Lastly, we will outline the three different training variations we will apply to the models.

3.1 Data

In this section we describe the data used for this project, and perform an initial, superficial, data analysis to see what we are working with. The data was generously provided by Mikael Johansson, professor at the Department of Psychology, Lund University.

It consists electrical brain activity measurements, in the form of EEG recordings from test-subjects who were exposed to visual and auditory stimuli.

More specifically, we have a pre-epoched dataset for each test-subject.[1] Each epoch consists of a time series of electrical activity over an approximately 9 second time period, during which various events take place, see event encoding in figure 3.1.

3.1.1 Metadata

The EEG data was recorded using a Neuroscan SynAmps RT (Compumedics) amplifier, Curry 7 software and 64 active Ag/AgCl electrodes mounted in an EasyCap, according to the 10% system, covering the 10/20 area, see figure 3.2. Data was recorded from 60 electrodes, these are all visible along with their associated names and positions from figure 3.2b, with a sampling rate of 1000 Hz downsampled to 500 Hz offline.

[1] Here epoch refers to a recording of EEG data, unlike in the context of neural networks.

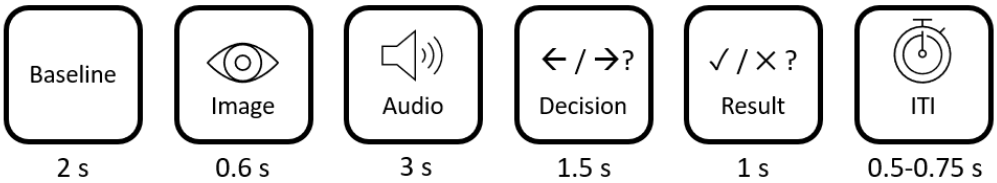


Figure 3.1: Encoding of each section of an epoch, the first being baseline recording, in the 2nd an image representing the word class is shown, in the 3rd is the word is played into either the subjects left or right ear, in the 4th the subject renders a decision on which ear they heard the word in, in the 5th the result is revealed, the last section is the inter-trial interval.

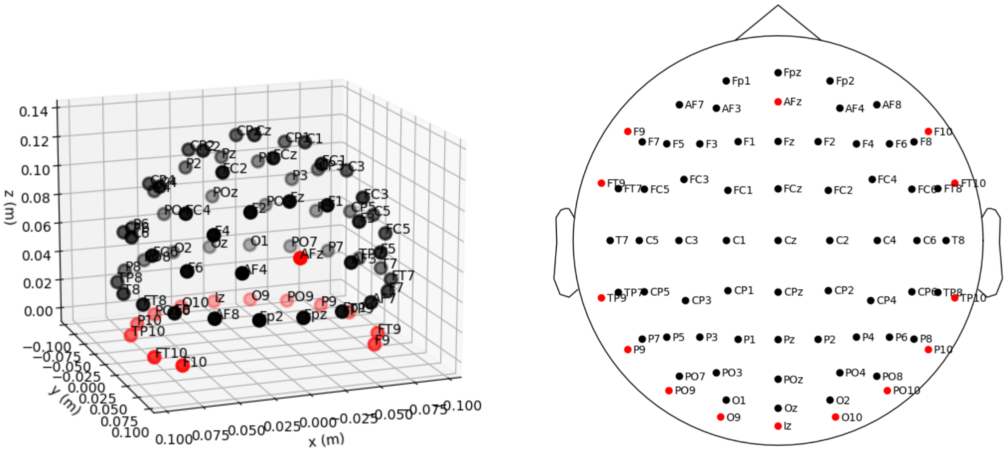
The words were presented auditorily in the experiment recorded in the same female voice, presented through in-ear headphones (SONY MDR EX650AP).

Data were pre-processed with a high-pass filter at 0.1 Hz and segmented into epochs of 9 seconds with event segments as encoded in figure 3.1. The data were baseline corrected based on the average of the whole epoch to minimize offsets. Line noise at 50 Hz was reduced and artifacts were removed by visual inspection. Independent component analysis (ICA) was applied to remove components corresponding to ocular and muscle artifacts.

Data were collected from 6 different test-subjects, with varying amount of epochs, see table 3.1.

Test Subject Info			
Subject	Epochs	#Left	#Right
Subject 10	352	174	178
Subject 11	343	169	174
Subject 12	366	182	184
Subject 13	368	184	184
Subject 14	349	179	170
Subject 15	370	186	184

Table 3.1: Metadata for each test-subject, amount of epochs for the test subject, how many of the epochs were labelled left and right. All subjects have balanced datasets, none are skewed heavily towards one label.



(a) Positions of EasyCap electrodes in 3D. With the FPz channel facing the same direction as the subjects nose.

(b) Positions of EasyCap electrodes in projected onto 2D cross-section of a subjects head. Triangle represents the nose.

Figure 3.2: Standard M1-EasyCap montage visualization with 74 channels, generated with the MNE library [14]. Obsolete channels are marked with red, leaving our 60 channels of interest with black markers.

3.1.2 Data Exploration

Each epoch contains a lot of information, as it consists of 60 channels, each of which is a time-series with 4501 data points. As it can be hard to visualize all 60 channels from one epoch, we start out with a single channel, see figure 3.3.

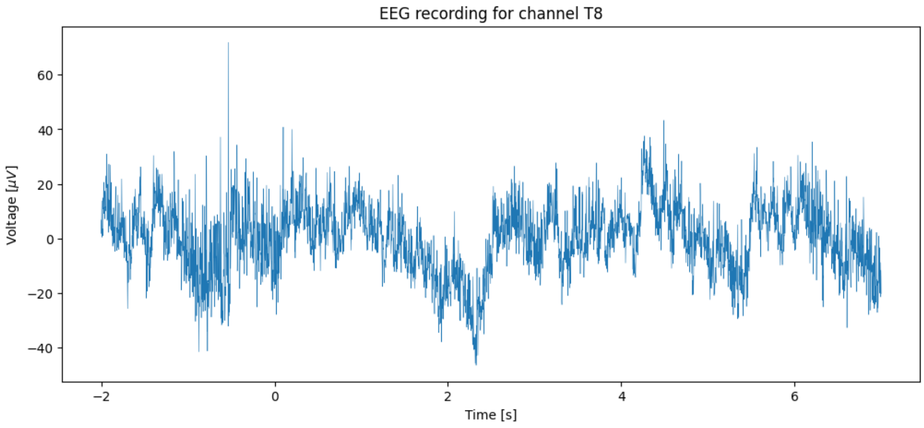


Figure 3.3: Channel T8 EEG recording from the first epoch from Subject 10.

Given the seemingly random/noisy data, it is hard to conclude anything from a graph like this, from our encoding in figure 3.1, we might expect, on average, to see distinct activity in each section, less during the baseline, more during visual and auditory stimuli etc.

We check this by averaging all epochs and creating a butterfly plot to visualize all channels, see figure 3.4.

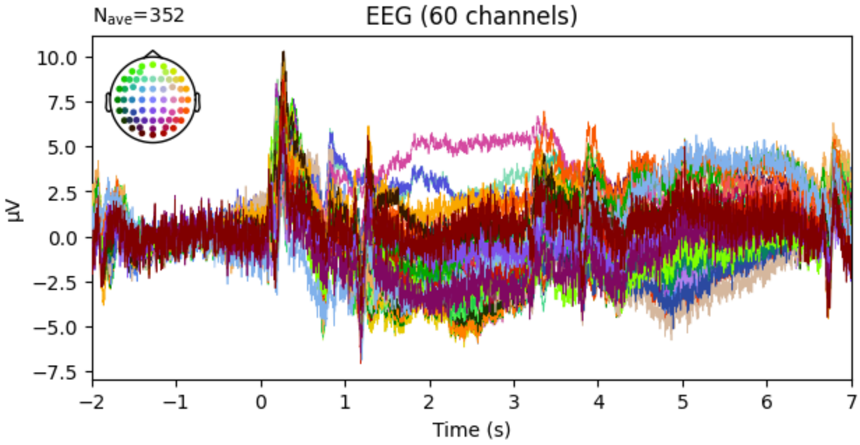


Figure 3.4: Butterfly plot depicting mean voltage for each EEG channel.

Despite averaging over all epochs, the butterfly plot still seems rather noisy, in an effort to make it easier to analyze, we perform a running mean over each channel to get a smoother plot, see figure 3.5.

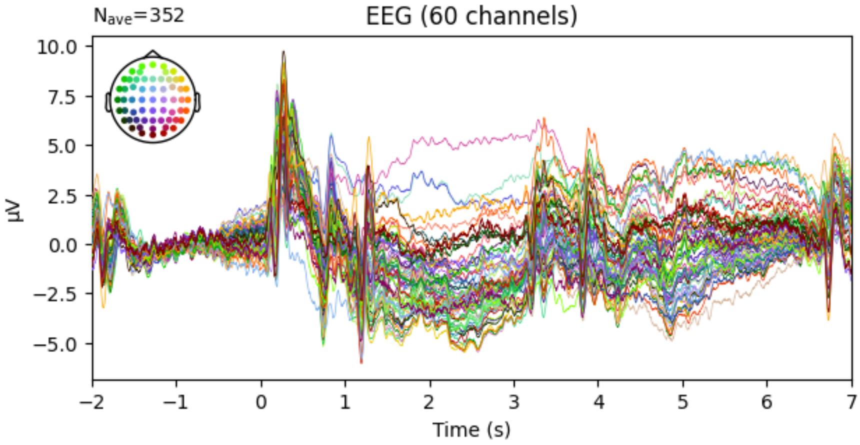


Figure 3.5: Same as figure 3.4, but with each channel smoothed.

From figure 3.5 the segments described in the encoding are clearly visible, we have low activity during the baseline period, a distinct spike during the visual stimulation, and then some kind of low voltage response during the auditory phase.

Since the auditory phase is split into two classes, namely whether the sound is played in the left, or right ear, we create a butterfly plot while averaging over each class, and inspect where activity is localized during time periods of interest, see figure 3.6 and 3.7.

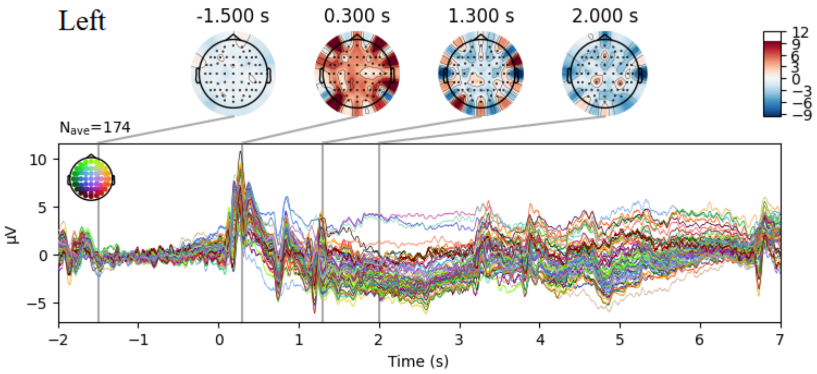


Figure 3.6: Smoothed butterfly plot depicting mean voltage for left-ear stimuli, with topographical activity maps of the scalp at points of interest.

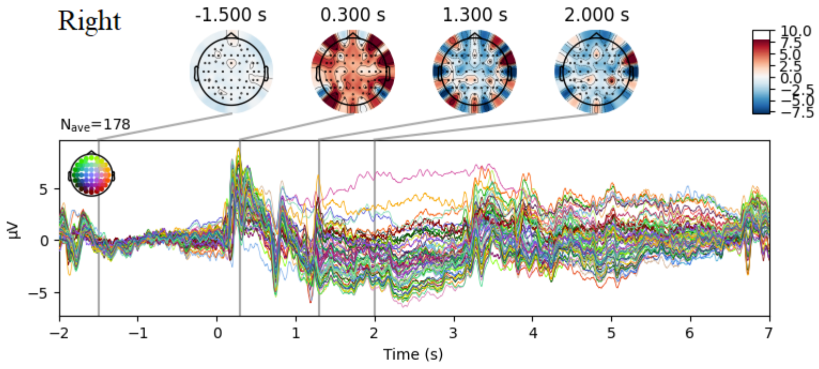


Figure 3.7: Smoothed butterfly plot depicting mean voltage for right-ear stimuli, with topographical activity maps of the scalp at points of interest.

As we can see, there is not much distinguishing the activity for right/left ear stimuli, so unfortunately we cannot make any conclusions about particularly interesting channels, although it does prove to be a welcome sanity check with regard to the encoding from figure 3.1.

3.2 Network Architectures

In this section, we detail the specific deep convolutional neural networks employed in this paper, namely EfficientNet V2 and ResNet18. Both of these networks were designed to perform well with image classification, specifically on the ImageNet dataset, but this easily generalizes to many other images through transfer learning.

3.2.1 EfficientNet V2

EfficientNet V2, henceforth referred to as EffNet, is amongst the newer architectures, specifically designed to achieve near state-of-the-art accuracy, but using significantly fewer computational resources [29].

EffNet is primarily constructed from the two block types, MBConv and Fused-MBConv, introduced alongside EffNet. The two blocks and their general structure can be seen from figure 3.8.

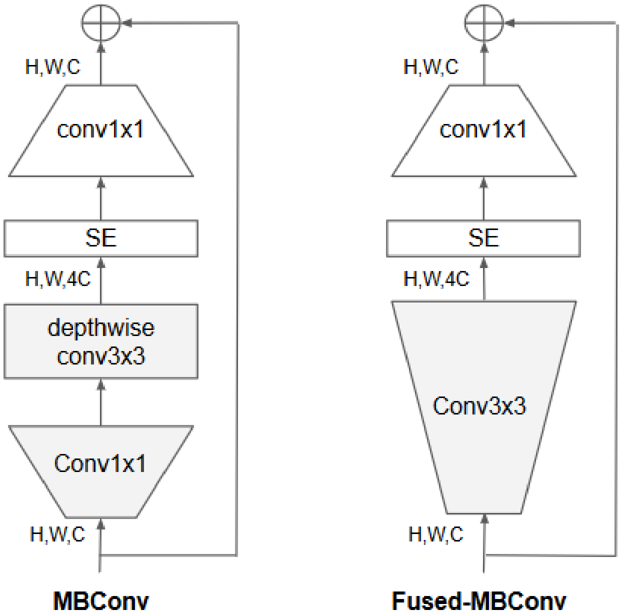


Figure 3.8: MBConv and Fused-MBConv, the primary building blocks of the EfficientNet V2 architecture [29].

The full architecture of EffNet, as given by the original authors, is given in table 3.2. From the blocks in figure 3.8 and overall structure in table 3.2 we gather that EffNet is a very large model. In fact, it contains over 21 million parameters, clearly it is not feasible to train this model on the modest data-set we have available.

Fortunately, the PyTorch library has the EffNet model, as well its weights when pre-trained on ImageNet available, so we will realistically be able to work with the model despite our limited training data [12].

Stage	Operator	Stride	#Channels	#Layers
0	Conv3x3	2	24	1
1	Fused-MBConv1, k3x3	1	24	2
2	Fused-MBConv4, k3x3	2	48	4
3	Fused-MBConv4, k3x3	2	64	4
4	MBConv4, k3x3, SE0.25	2	128	6
5	MBConv6, k3x3, SE0.25	1	160	9
6	MBConv6, k3x3, SE0.25	2	256	15
7	Conv1x1 & Pooling & FC	-	1280	1

Table 3.2: The EfficientNet V2 architecture, where #Layers describes the amount of occurrences of the operator in the associated stage. From #Channels we see the final stage has 1280 features going into the fully connected layer [29]. Here stride refers to the filter's ”step size” during convolution.

3.2.2 ResNet18

ResNet18, henceforth referred to as ResNet, is a well known DNN, also specialized in image classification. Like EffNet, ResNet is a large model with over 11 million parameters, and is available as a pre-trained model from PyTorch [13].

Like EffNet, ResNet is also made up of simpler blocks, named BasicBlocks. The structure of BasicBlocks can be seen from figure 3.9.

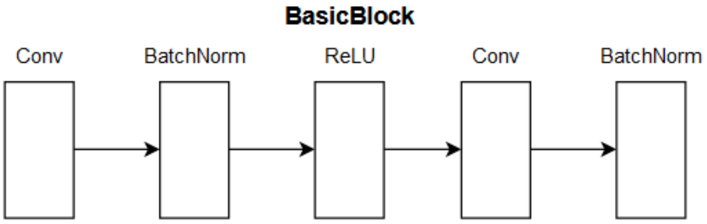


Figure 3.9: The general structure of the basic block, notably it can include a downsampling section.

The full architecture, although simplified from the original version, can be seen in table 3.3. Clearly it is still a large network, but significantly smaller than EffNet, both when considering total parameters, and amount of layers. Thus, we expect shorter training times for ResNet.

Stage	Operator	Stride	#Channels	#Layers
0	Conv7x7 & MaxPool3x3	2	64	1
1	BasicBlock, Conv3x3	1	64	2
2	BasicBlock, Conv3x3	-	128	2
3	BasicBlock, Conv3x3	-	256	2
4	BasicBlock, Conv3x3	-	512	2
5	AvgPool, FC	-	512	1

Table 3.3: The ResNet18 architecture, where #Layers describes the amount of occurrences of the operator in the associated stage. From #Channels we see the final stage has 512 features going into the fully connected layer. The BasicBlock in layer 1 of stage 2-4 includes a downsampling [16].

3.3 Approach

In this section we detail how we prepared the data for model training, and how exactly we structured the model training and modifications to get our initial results.

It is worth mentioning, that a general approach is taken, in order to determine which method seems to have the most promise. Subsequently, more attention can be given to a single model, with the goal of improving performance.

3.3.1 Pre-Processing

First of all, we decimate the EEG sampling frequency to 64Hz, in order to reduce computational cost. This is done under the assumption that we are only interested in the 0-30Hz region of the time series data, as this is where the conventionally interesting brainwaves lie [1]. The decimation is done with the class method `Epochs.resample` from the MNE library [14].

Both ResNet and EffNet are designed for image classification, and therefore only take images as inputs (or at least anything in the same format). Specifically, EffNet takes 384x384 images in RGB format, meaning we need an additional dimension to account for each of the three color channels, so EffNet takes tensors of shape 384x384x3.[2] Similarly, for ResNet we need tensors of shape 224x224x3.

The simplest way to accommodate this, is by creating images from the EEG data to fit these formats. Here we choose to go with time-frequency representations (TFR) of the EEG data, as this format has shown promise within EEG classification using CNNs [19]. The input resolution difference between ResNet and EffNet can be dealt with by resizing the image to fit, using the PyTorch transform `torchvision.transforms.Resize`.

To obtain our TFR representations we use a Morlet wavelet transform as computed by the function `time_frequency.tfr_morlet` from the Python library MNE. This method, as a spectral analysis approach, is formally equivalent to that of a short-time Fourier transform, so we will not concern ourselves too much with the choice of TFR method [4].

Furthermore, we log-transform the wavelet-transformed results, in order to prevent the lower frequencies from dominating the visual, as the EEG data contains pink noise, which can be dealt with using a log-transform [6]. This is done to ensure interpretability of the TFR in the entire frequency range of interest, and not only have a useful visual near the lower end of the frequency range.

The TFR is simply a matrix representing the signal power for each frequency and time index. So it doesn't immediately fit the RGB format, the TFR matrix can simply be plotted with a chosen color mapping to yield an RGB formatted image, see example in figure 3.10. All TFR images represent the frequency range 1-30Hz on the y-axis and time range of 2.6-5.6 seconds on the x-axis, as this is when the auditory stimuli occurs.

Though, here we are mapping the one-dimensional power to the three-dimensional space of RGB values. In principle, when ignoring the specifics of transfer learning, there should not be any gained information here, as compared to a single-channel grayscale image, only the RGB version fits the input format.

We put this to the test, by also generating a different kind of TFR image, namely three grayscale TFRs from the channels T7, Cz, T8, and stack them into a single RGB formatted image, see figure 3.11, this image-type will henceforth be referred to with the GC3 (Gray-Channel-3). This method has also seen previous success [31]. The regular colormap representations are created from channel T8, and will just be referred to as RGB images.

[2] Not necessarily in this order, but that is a technicality only relevant in the code.

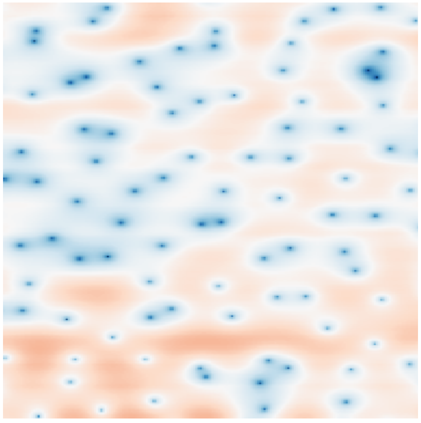


Figure 3.10: An RGB color-mapped Morlet-wavelet transform TFR of channel T8 from subject 10. Frequency ranging from 1-30Hz on the y-axis, time from 2.6-5.6 seconds on the x-axis.

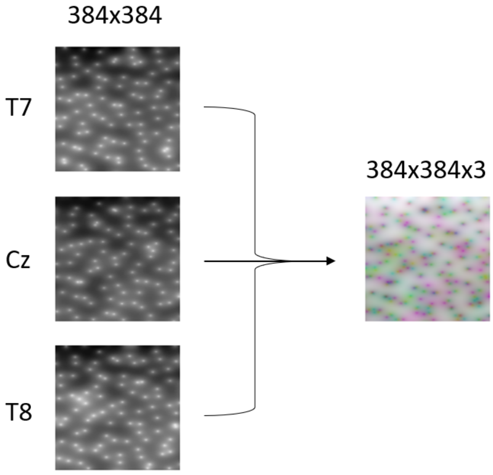


Figure 3.11: Visual of GC3 construction method, generates gray-scale TFR for channels T7, Cz and T8, and stacks them into a single RGB image.

3.3.2 Intra-Subject

For intra-subject modelling, we train and evaluate the models only on a single subject. We use only Subject10, due to the more extensive pre-processing. We create an 80/20 train/test split, and train the models for 100 epochs.

For both ResNet and EffNet we use the hyperparameters described in table 3.4. Here all parameters are PyTorch training parameters, batch size is how many training examples are processed simultaneously, criterion is the choice of loss function, optimizer is the choice of technique to minimize the empirical risk (SGD is stochastic gradient descent), and the scheduler is a modifier that alters the learning rate of the optimizer. More details can be found in the PyTorch library documentation [25].

Training is performed in Google Colab, using their freely provided GPUs, no exact specifications are provided, since they can vary over time depending on availability. Seemingly, there is 12GB of GPU RAM available, as we run out of memory when exceeding this amount. The computation environment and constraints are the same for all model training discussed in this project.

Parameter	Value
Batch Size	4
Criterion	CrossEntropyLoss
Optimizer	SGD
Learning Rate	0.001
Momentum	0.09
Scheduler	StepLR
Step Size	7
Gamma	0.1

Table 3.4: Notable PyTorch training hyper parameters for the intra-subject model training. The indented parameters are parameters of their parent.

We perform 3 varieties of training for each of the model, TFR-type combinations:

- Full Train
  Training of the full model, with no pre-trained weights. This will almost certainly either overfit, or fail to learn anything at all, but serves as a good reference point.

- Full Tune
  Training of the full model, with pre-trained weights, i.e. using the weights as initialized by the training on ImageNet and adapting to the new dataset.

- Fine Tune
  Freeze entirety of model, except the last fully connected layer which classifies left/right.

3.3.3 Inter-Subject

For inter-subject modelling, we train the model on Subject11-Subject15 and use Subject10 for validation. Here we investigate the models' ability to generalize between subjects.

We train with the leave out Subject10 split, for 50 epochs.[3] We use the same hyper parameters as in table 3.4, with the exception of batch size, which we upped to 8 to increase computation speed whilst staying below the RAM limit.

Again, we perform 3 varieties of training for each mode, TFR-type combination:

- Tune Full
  Training of the full model, with pre-trained weights, i.e. using the weights as initialized by the training on ImageNet and adapting to the new dataset.
- Tune FC
  Freeze entirety of model, except the last fully connected layer which classifies left/right.
- Tune Block
  Freeze section of the model, leaving a block of the final section with re-initialized trainable weights, which are then all trained to optimize the left/right classification.

Really, only the Tune Block training results are of interest for further model development here, the two first training variants is to establish a reference point. The goal here is to take advantage of the pre-trained weights in the beginning of the network, which are presumably apt at general purpose image analysis feature extraction, and tune the later weights to the more specific task of EEG classification. The new weights early in the block should then be proficient in extracting useful features for EEG classification, and the later weights can then be tuned to a specific task, that is what we will try in the next section.

For Tune Block, we need to select a cutoff for each network, from which to make the weights trainable. For EffNet, we let the entirety of stage 7 from table 3.2 be trainable. For ResNet we let the 2nd layer of stage 4 be trainable, although with the last batch norm replaced with a dropout layer,

[3] 100 epochs turned out to be very excessive, given the amount of data, even 50 is more than enough it seems.

furthermore we let stage 5 be trainable, but with a dropout layer before the fully connected layer. Both dropout layers have $p = 0.5$.

3.3.4 Inter-Subject Transfer Learning

For inter-subject transfer learning we combine the two previous methods. We first train and validate on Subject11-Subject15, here in an attempt to pre-train the models on a larger body of subjects, and thereby adapt the weights to not just feature extract from regular images, but specifically EEG TFR images, we refer to this as the *inter phase*. Subsequently, we use the updated model to perform transfer learning and tune the model to perform well on subject 10, like in the intra-subject modelling section, referred to as the *intra phase*.

Seeing the previous results, we will only continue with the best amongst the RGB and GC3 data, in order to focus on the highest potential TFR. We will train one model with ResNet and one with EffNet, both will be trained in two phases as described above, first we tune a large block with inter subject tuning, after which we perform intra-subject tuning on a smaller block. To see exactly which model parameters will be trainable during each phase, see table 3.5.

Trainable Parameters		
Phase	ResNet	EffNet
Inter	Stage 4 Stage 5	Stage 6, 15^{th} layer Stage 7
Intra	Stage 4, 2^{nd} layer Stage 5	Stage 7 -

Table 3.5: Parameters in each model that remains trainable during each training phase, the rest of the parameters are frozen. See table 3.3 and 3.2 to see the architecture components referred to.

Automated hyperparameter optimization is not feasible with the resources available, since Google Colab will stop the runtime if the session lasts too long, or is idle. So the hyperparameters are adjusted manually, our choice of hyperparameters for each model can be seen in table 3.6 and 3.7. Note, for ResNet, we add a dropout layer before the fully connected layer, the dropout probability is noted in table 3.6.

ResNet Hyperparameters

Parameter	Inter	Intra
Batch Size	8	8
Criterion	CrossEntropyLoss	CrossEntropyLoss
Optimizer	SGD	SGD
Learning Rate	0.001	0.001
Momentum	0.09	0.09
Weight Decay	5e-2	-
Scheduler	StepLR	StepLR
Step Size	7	7
Gamma	0.1	0.1
Dropout	0.5	0.8

Table 3.6: Notable PyTorch training hyper parameters for each phase of the ResNet model training. The indented parameters are parameters of their parent.

EffNet Hyperparameters

Parameter	Inter	Intra
Batch Size	8	8
Criterion	CrossEntropyLoss	CrossEntropyLoss
Optimizer	Adam	SGD
Learning Rate	0.001	0.001
Momentum	-	0.09
Weight Decay	1e-1	0.5
Scheduler	StepLR	StepLR
Step Size	7	7
Gamma	0.1	0.1

Table 3.7: Notable PyTorch training hyper parameters for each phase of the EffNet model training. The indented parameters are parameters of their parent.

Chapter 4

Results and Discussion

In this chapter we present the training results for the three training methods described for EffNet and ResNet, and discuss their potential implications on transfer learning for EEG classification.

The final classifiers, trained according to the Inter-Subject Transfer Learning approach, are evaluated in order to determine whether the result is significantly different from the trivial classifier. This is done using McNemar's test.

4.1 Intra-Subject

In this section, we test the performance of ResNet and EffNet as intra-subject classifiers on both the RGB and GC3 time-frequency representations of the EEG recordings from Subject 10.

We compare 3 different training methods for intra-subject, namely, training the full model from scratch (Full Train), tuning the full pre-trained model (Full Tune) and training only the last fully connected layer (Fine Tune).

The training and validation results for each epoch can be seen in figure 4.1 and 4.2 for EffNet and ResNet respectively. Unsurprisingly, none of them yield particularly impressive accuracies, but it is worth noting, that ResNet seemingly is more prone to overfitting in general, as seen from the gap between training and validation accuracy for all the 3 training methods.

We have summarized our results in table 4.1, we include the mean as an interesting metric, since the accuracies from figure 4.1 and 4.2 were rather unstable. In principle, we should just pick the best model we find throughout training, hence the max accuracy.[1]

[1] With the condition that validation accuracy is not much larger than training accuracy.

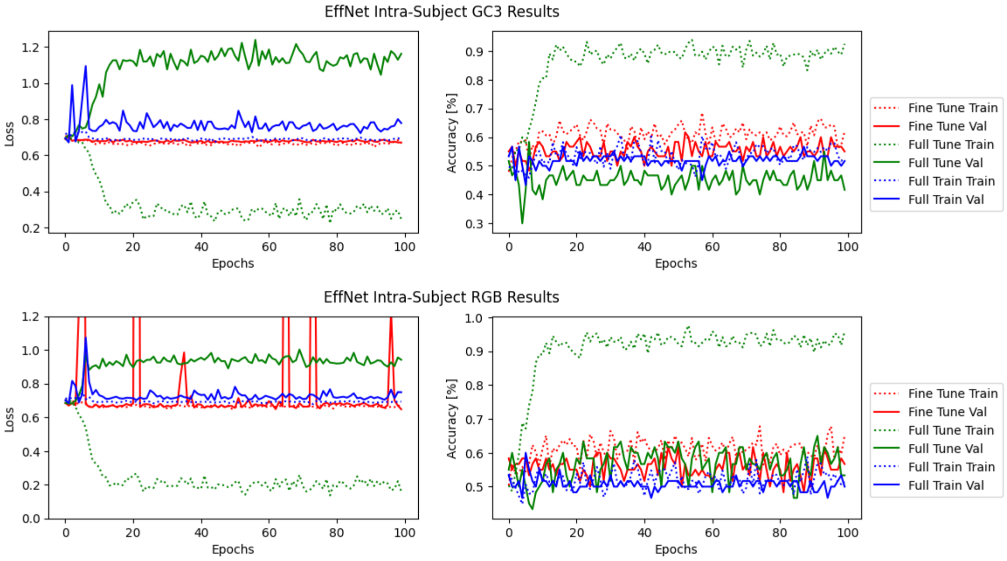


Figure 4.1: Intra-subject loss and accuracy results from 3 training methods for EffNet on the GC3 (top) and RGB (bottom) TFR variants of Subject 10's EEG recordings. The RGB loss has very large spikes, the y-axis is capped to preserve interpretability.

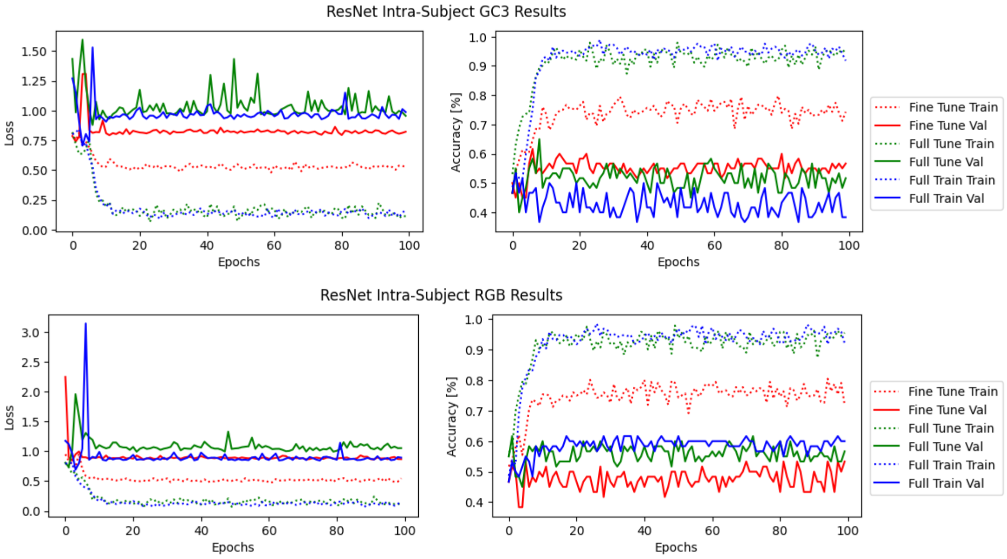


Figure 4.2: Intra-subject loss and accuracy results from 3 training methods for ResNet on the GC3 (top) and RGB (bottom) TFR variants of Subject 10's EEG recordings.

Intra-Subject Accuracies					
Model	Method	RGB		GC3	
		Max	Mean	Max	Mean
ResNet18	Full Train	0.617	0.584	0.533	0.430
	Full Tune	0.617	0.559	0.650	0.518
	Fine Tune	0.533	0.476	0.617	0.551
EffNet V2	Full Train	0.600	0.507	0.567	0.520
	Full Tune	0.650	0.566	0.583	0.453
	Fine Tune	0.617	0.558	0.617	0.553

Table 4.1: Overview of intra-subject validation accuracy from 3 different training methods on ResNet18 and EfficientNet V2 with the RGB and GC3 versions of the EEG TFR. The maximum validation accuracy over all epochs is taken, as well as the mean.

From table 4.1, one might notice that the results for the GC3 TFR generally are worse than for RGB. From a transfer learning perspective, this makes sense, due to the source domain (the domain representing the ImageNet images), $\mathcal{D}_{ImageNet}$, being more closely related to $\mathcal{D}_{RGB}$ than $\mathcal{D}_{GC3}$.

The pre-trained model is specialized in picking out features from images representing regular images from the real world, so naturally our TFR images are further removed, as they are constructed images. Even then, at least in the RGB image, all the color channels collectively represent somewhat natural shapes, just like for ImageNet. On the contrary, GC3 represents distinct, somewhat natural, shapes in each color channel, the same cannot be said for the GC3 images as a whole.

As it turns out, the additional information contained in the GC3 images do not outweigh the negative effect of being part of a more alien target domain, despite intuitively having a higher potential for learning. Unfortunately, this potential was never realized with the chosen architecture and pre-training.

Notably, from the 4.2, we see that for RGB, full tune and full train are largely equivalent, suggesting that the pre-trained weights had little to no impact. Although, this is not the case for EffNet.

4.2 Inter-Subject

In this section we investigate how well our DNN's can generalize between subjects when trained on subjects 11 through 15, and tested on subject 10.

We compare 3 different training methods for inter-subject, namely, tuning the full pre-trained model (Tune Full), tuning the last FC layer (Tune FC) and tuning a block at the end of the network (Tune Block)

The results for ResNet and EffNet can be seen from figures 4.3 and 4.4 respectively. The max and mean accuracies can be seen from table 4.2.

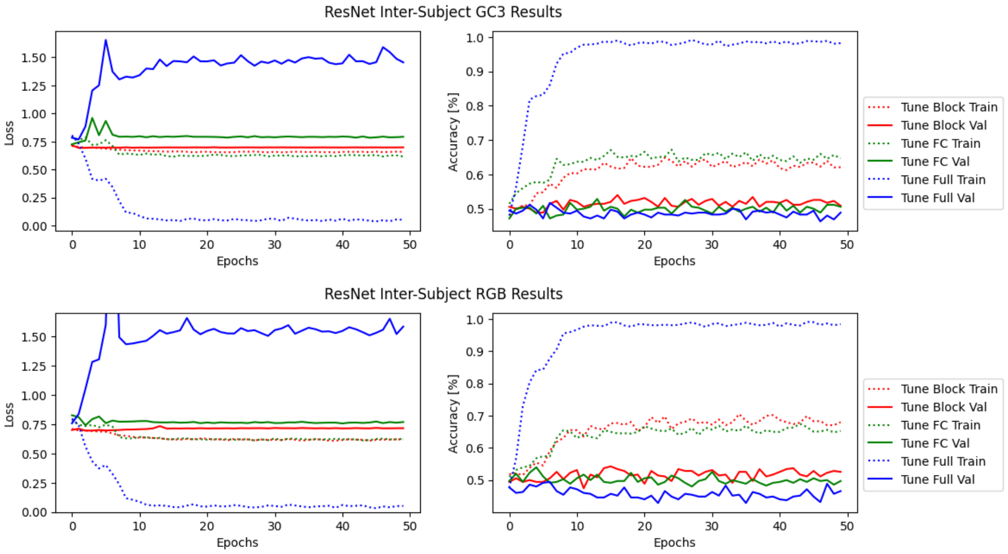


Figure 4.3: Inter-subject loss and accuracy results from 3 training methods for ResNet on the GC3 (top) and RGB (bottom) TFR variants of test-subject EEG recordings. Trained on Subject 11-15, tested on Subject 10.

Clearly, Tune Full overfitted in every case, which is no surprise, given the massive amount of trainable parameters. Tune Block was the highest performer in every case, it seems the difference between source and target domains are too large for us to rely on fixed feature extraction. We should allow for the network to partially adapt to the target domain, after which we see better classification results.

Again, GC3 underperformed when comparing to RGB, it seems the trend of poor generalization continues.

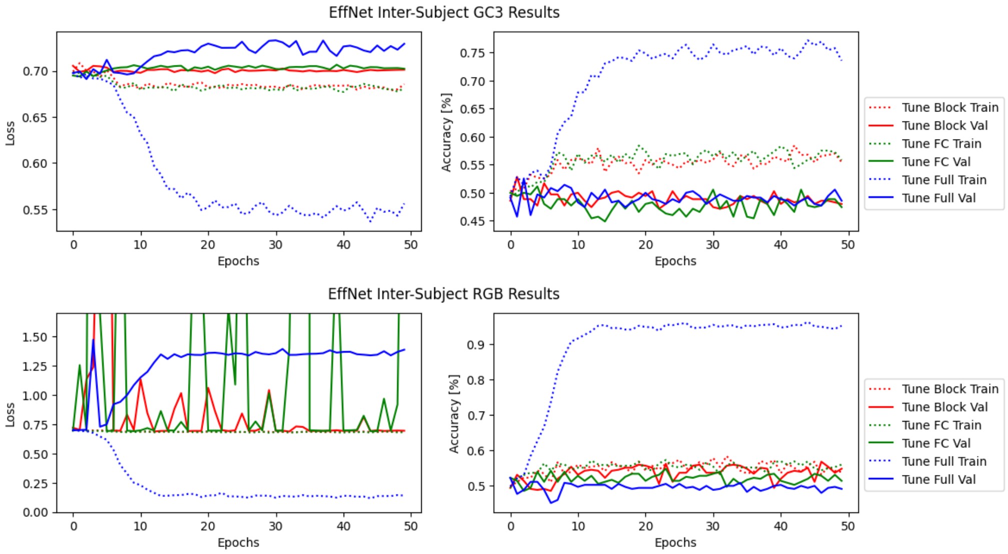


Figure 4.4: Inter-subject loss and accuracy results from 3 training methods for EffNet on the GC3 (top) and RGB (bottom) TFR variants of test-subject EEG recordings. Trained on Subject 11-15, tested on Subject 10. The RGB loss has very large spikes, the y-axis is capped to preserve interpretability.

Inter-Subject Accuracies					
Model	Method	RGB		GC3	
		Max	Mean	Max	Mean
ResNet18	Tune Full	0.494	0.458	0.517	0.486
	Tune FC	0.540	0.500	0.528	0.499
	Tune Block	0.543	0.516	0.540	0.516
EffNet V2	Tune Full	0.523	0.496	0.526	0.490
	Tune FC	0.548	0.521	0.511	0.478
	Tune Block	0.568	0.536	0.528	0.489

Table 4.2: Overview of inter-subject validation accuracy from 3 different training methods on ResNet18 and EfficientNet V2 with the RGB and GC3 versions of the EEG TFR. The maximum validation accuracy over all epochs is taken, as well as the mean.

4.3 Inter-Subject Transfer Learning

In this section, we investigate whether there are improvements to be made by pre-training the model as described in the Inter-Subject section, and then fine tuned on subject 10.

From the previous results, the models generally achieve higher accuracies with the RGB data, this is particularly evident from the mean accuracy in table 4.1 and 4.2. The training results for the new Inter and Intra-Phase can be seen from figure 4.5.

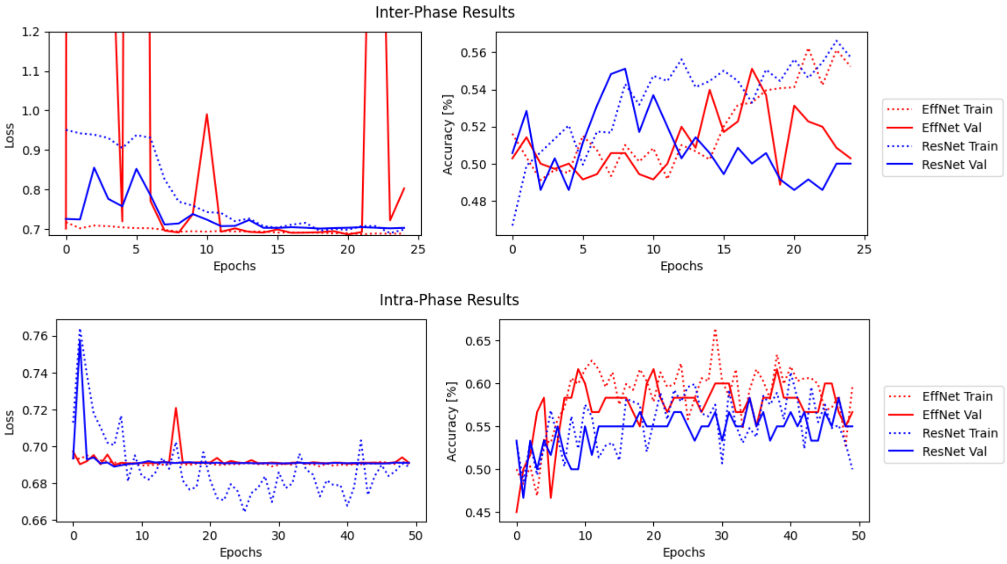


Figure 4.5: Results from Inter and Intra-Phase for both ResNet and EffNet trained on the RGB data.

In the inter-phase seen in figure 4.5, no model is especially convincing, their validation accuracies do not have a general upwards trend. On the contrary, in the intra-phase EffNet has a consistent edge on ResNet, although not by much.

The validation accuracies are summarized for each model/phase combination in table 4.3. Here it is also evident that EffNet has performed the best.

Now, seeing as the accuracies are all above 50%, one might think that means the models have learned something useful, this should be tested. The models are performing binary classification, and both models have computationally intensive training phases, making cross validation impractical, in this case McNemar's test is an appropriate choice [8].

Inter-Subject Transfer Learning Accuracies			
Phase	Model	Max	Mean
Inter	ResNet	0.551	0.508
	EffNet	0.551	0.511
Intra	ResNet	0.583	0.545
	EffNet	0.617	0.572

Table 4.3: Inter-Subject Transfer Learning maximum and mean validation accuracy for ResNet and EffNet over the 50 training epochs for both the inter and intra-subject training phase.

We compare our models to the trivial classifier, namely the classifier that always picks the most common label, we do this to determine whether our classifiers performs significantly better than chance level. Specifically, we will test the null hypothesis that our classifier is the same as the trivial classifier.

To perform McNemar's test, we first set up a contingency table for correctly/incorrectly classified results by our classifier and the trivial classifier, see table 4.4.

		Trivial	
		T	F
EffNet	T	25	12
	F	7	16

		Trivial	
		T	F
ResNet	T	13	22
	F	19	6

Table 4.4: Contingency table for both EffNet (left) and ResNet (right), each compared to the trivial classifier. T denotes a true classification, F denotes a false classification. Meaning for EffNet, we have 25 validation examples correctly classified by both EffNet and the trivial classifier, and 12 that were correctly classified by EffNet, but not by the trivial classifier etc.

Using the discordant pair (i.e. values on the anti-diagonal, so values where classifier predictions disagree), let the anti-diagonal values be denoted a and b (order doesn't matter), and we can calculate the continuity corrected McNemar test statistic

$$T = \frac{(|a-b|-1)^2}{a+b} \sim \chi_1^2,$$

which is distributed according to χ^2 with 1 degree of freedom [8]. Knowing this we can calculate the test statistic, T, and p-value for each test, see the results in table 4.5.

Model	T	p
ResNet	0.098	0.755
EffNet	0.842	0.359

Table 4.5: McNemar continuity corrected test statistic and associated p-value for the McNemar test performed on both ResNet and EffNet compared with the trivial classifier.

From table 4.5, we get that, according to McNemar's test, neither the EffNet or ResNet trained classifiers are significantly different from the trivial classifier. So the results here are not exactly convincing, there is insufficient certainly to claim anything much about our classifiers. We would need either higher accuracy or a larger validation sample to potentially get sufficient discordant values to result in a significant test statistic.

Even without considering the failure to reject the null hypothesis, the Inter-Subject Transfer Learning failed to result in higher accuracy than the Intra-Subject training, from table 4.1 we see accuracies of 0.650.

Chapter 5

Conclusion

We achieved a maximum inter-subject transfer learning accuracy with EfficientNet V2 of 61.7%. We were able to create multiple models with accuracies around 60%, although the most carefully trained and tested classifiers, trained with Inter-Subject Transfer Learning were not significantly different from the trivial classifier according to McNemar's test.

We see no evidence of quasi-inductive transfer learning being effective in this case, as the pre-trained weights did not consistently improve the classification results notably, when compared to a full training of the same model.

We end this project without a significant result, but with some strong notions about properties of transfer learning. Seemingly, the GC3 data performed worse overall than the RGB data, which from a transfer learning perspective makes sense, as the source domain of the pre-trained model, ImageNet, shares more characteristics with the RGB data. Thus, the pre-trained weights should transfer better to the RGB target domain, in spite of the GC3 data containing more information.

More generally, the value of the pre-trained weights is unclear, there are many improvements that could be made to yield more convincing results, but more interestingly there is an entire avenue of other studies that could be made with pre-trained classifiers that have been trained in a source domain similar to that of the target.

5.1 Improvements

In this study, we made a variety of assumptions, some seemed to hold up perfectly fine, such as the log-transform of the power, since EEG contains 1/f noise [6]. On the contrary, our assumption that only frequencies in the

0-30Hz range are of interest may, in hindsight, have been a mistake. Previous literature suggests, auditorily triggered EEG responses lie in the range of 40-150Hz [22]. Thus, we might inadvertently have excluded the region of interest from our analysis, although this is far from certain. A potential improvement would then be, to update the region of interest for the computed TFRs.

Another low-hanging fruit, with regard to improvements, would be to perform cross validation for each model training segment. This was not a guaranteed possibility due to the computational limitations of the freely available Google Colab GPUs [26]. Provided with more compute, it would be simple to set up more comprehensive cross validation and hyperparameter tuning, which likely would yield better results.

Hyperparameter tuning is another area in which improvements would likely be achieved, this is a broad area, with many parameters on which to optimize: choice of channel, optimizer, learning rate, scheduler, batch size, and so many more.

Additionally, the type of data augmentation and transforms included could also be optimized, unfortunately in this project, we never managed to perform data augmentation due to limitations in available computational resources, this would also be a strong contender amongst improvement opportunities.

Furthermore, there is the more specific issue that we are trying to determine whether a sound was heard from the left and the right, but the test subject is also meant to determine from which direction he/she heard something. This means we do not know whether our model classifies which direction a sound was heard from, or which direction the subject is thinking about, since it is likely that the subject thought about this before the audio phase was over. Statistical tests could potentially be performed to investigate this.

5.2 Further Research

From the literature exploration associated with the project, it seems to be the most promising models use raw EEG data as input. With the amount of parameters in these deep neural networks it is really no surprise, they are capable of learning patterns with complexities beyond the ability of humans to interpret. With the raw EEG data, you do not trade potentially useful information for interpretability, like we do with the TFR transforms. It has been shown that Deep CNNs are able to take advantage of this [5].

Perhaps even more interestingly, the recently popularized transformer architectures used for NLP have also shown promise within the domain of

EEG classification, notably the BENDR model network has been able to generalize to many downstream classification tasks using transfer learning [18].

Given the rapid advancement of image classification with DNNs spurred on by ImageNet, and their impressive aptitude for transfer learning, a similar approach would be interesting for BCI. The creation of a large dataset, akin to ImageNet, with raw EEG data on a unified format from different subjects, locations, ages, recording equipment etc., would allow researchers, and even Kaggle-enthusiasts put their skills to the test and push the limits BCI technology. Even if it starts with a single model like BENDR, making the pre-trained model easily available on a site like Hugging Face, could very well result in a surge of new results within the field.

There has also been an interesting recent development within the subfield of liquid networks, networks whose parameters are able to change to adapt to the incoming data stream, even after the training phase. These networks are constructed to even more closely resemble real neurons and neuron interactions than regular ANNs. Recently, the closed form solution to the differential equation describing neuron interactions has been found by researchers at MIT, this has already led to a significant increase in the speed, and viability of these liquid networks [15].

Perhaps the use of networks that more closely resemble our own biology, will be able to better classify biological data, EEG data might be a good candidate, since it arises directly from neurons firing.

References

[1] Priyanka A. Abhang, Bharti W. Gawali, and Suresh C. Mehrotra. "Chapter 2 - Technological Basics of EEG Recording and Operation of Apparatus". In: *Introduction to EEG- and Speech-Based Emotion Recognition*. Ed. by Priyanka A. Abhang, Bharti W. Gawali, and Suresh C. Mehrotra. Academic Press, 2016, pp. 19–50. ISBN: 978-0-12-804490-2. DOI: https://doi.org/10.1016/B978-0-12-804490-2.00002-6. URL: https://www.sciencedirect.com/science/article/pii/B9780128044902000026.

[2] Alexander Amini and Ava Soleimany. *Introduction to Deep Learning*. URL: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf. (accessed: 20.11.2022).

[3] Sunitha Basodi et al. "Gradient amplification: An efficient way to train deep neural networks". In: *Big Data Mining and Analytics* 3.3 (2020), pp. 196–207. DOI: 10.26599/BDMA.2020.9020004.

[4] Andreas Bruns. "Fourier-, Hilbert- and wavelet-based signal analysis: are they really different approaches?" In: *Journal of Neuroscience Methods* 137.2 (2004), pp. 321–332. ISSN: 0165-0270. DOI: https://doi.org/10.1016/j.jneumeth.2004.03.002. URL: https://www.sciencedirect.com/science/article/pii/S0165027004001098.

[5] Yücel Çimtay and E. Ekmekcioglu. "Investigating the Use of Pretrained Convolutional Neural Network on Cross-Subject and Cross-Dataset EEG Emotion Recognition". In: *Sensors* 20 (Apr. 2020). DOI: 10.3390/s20072034.

[6] S. Dave, Trevor Brothers, and Tamara Swaab. "1/ f Neural Noise and Electrophysiological Indices of Contextual Prediction in Aging". In: *Brain Research* 1691 (Apr. 2018). DOI: 10.1016/j.brainres.2018.04.007.

[7] Tim Dettmers. *Understanding Convolution in Deep Learning*. URL: https://timdettmers.com/2015/03/26/convolution-deep-learning/. (accessed: 21.11.2022).

[8] Thomas G. Dietterich. “Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms”. In: *Neural Computation* 10.7 (Oct. 1998), pp. 1895–1923. ISSN: 0899-7667. DOI: 10.1162/089976698300017197. eprint: https://direct.mit.edu/neco/article-pdf/10/7/1895/814002/089976698300017197.pdf. URL: https://doi.org/10.1162/089976698300017197.

[9] IBM Cloud Education. *Convolutional Neural Networks.* URL: https://www.ibm.com/cloud/learn/convolutional-neural-networks. (accessed: 21.11.2022).

[10] IBM Cloud Education. *Neural Networks.* URL: https://www.ibm.com/dk-en/cloud/learn/neural-networks. (accessed: 20.11.2022).

[11] IBM Cloud Education. *Overfitting.* URL: https://www.ibm.com/cloud/learn/overfitting. (accessed: 22.11.2022).

[12] The PyTorch Foundation. *EFFICIENTNET_V2_S.* URL: https://pytorch.org/vision/main/models/generated/torchvision.models.efficientnet_v2_s.html. (accessed: 4.12.2022).

[13] The PyTorch Foundation. *RESNET18.* URL: https://pytorch.org/vision/main/models/generated/torchvision.models.resnet18.html. (accessed: 4.12.2022).

[14] Alexandre Gramfort et al. “MEG and EEG data analysis with MNE-Python”. In: *Frontiers in Neuroscience* 7 (2013), p. 267. ISSN: 1662-453X. DOI: 10.3389/fnins.2013.00267. URL: https://www.frontiersin.org/article/10.3389/fnins.2013.00267.

[15] Ramin Hasani et al. “Closed-form continuous-time neural networks”. In: *Nature Machine Intelligence* (Nov. 2022). ISSN: 2522-5839. DOI: 10.1038/s42256-022-00556-7. URL: https://doi.org/10.1038/s42256-022-00556-7.

[16] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[17] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. URL: https://www.sciencedirect.com/science/articl/pii/0893608089900208.

[18] Demetres Kostas, Stéphane Aroca-Ouellette, and Frank Rudzicz. "BENDR: Using Transformers and a Contrastive Self-Supervised Learning Task to Learn From Massive Amounts of EEG Data". In: *Frontiers in Human Neuroscience* 15 (2021). ISSN: 1662-5161. DOI: 10.3389/fnhum.2021.653659. URL: https://www.frontiersin.org/articles/10.3389/fnhum.2021.653659.

[19] Quan Liu et al. "Spectrum Analysis of EEG Signals Using CNN to Model Patient's Consciousness Level Based on Anesthesiologists' Experience". In: *IEEE Access* 7 (2019), pp. 53731–53742. DOI: 10.1109/ACCESS.2019.2912273. URL: https://ieeexplore.ieee.org/document/8695791.

[20] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. Adaptive Computation and Machine Learning. Cambridge, MA: MIT Press, 2018. 504 pp. ISBN: 978-0-262-03940-6.

[21] Santiago Morales and Maureen E Bowers. "Time-frequency analysis methods and their application in developmental EEG data". In: *Developmental cognitive neuroscience* (Apr. 2022). ISSN: 2522-5839. DOI: 10.1016/j.dcn.2022.101067. URL: https://doi.org/10.1016/j.dcn.2022.101067.

[22] Suresh Muthukumaraswamy. "High-frequency brain activity and muscle artifacts in MEG/EEG: A review and recommendations". In: *Frontiers in human neuroscience* 7 (Apr. 2013), p. 138. DOI: 10.3389/fnhum.2013.00138.

[23] Michael Nielsen. *How the backpropagation algorithm works*. URL: http://neuralnetworksanddeeplearning.com/chap2.html. (accessed: 20.11.2022).

[24] Sinno Jialin Pan and Qiang Yang. "A Survey on Transfer Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2010), pp. 1345–1359. DOI: 10.1109/TKDE.2009.191.

[25] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *CoRR* abs/1912.01703 (2019). arXiv: 1912.01703. URL: http://arxiv.org/abs/1912.01703.

[26] Google Research. *Colaboratory*. URL: https://research.google.com/colaboratory/faq.html. (accessed: 05.12.2022).

[27] J. J. Shih, D. J. Krusienski, and J. R Wolpaw. “Brain-computer interfaces in medicine”. In: *Mayo Clinic proceedings* (Mar. 2012). DOI: 10.1016/j.mayocp.2011.12.008. URL: https://doi.org/10.1016/j.mayocp.2011.12.008.

[28] Pavithra Solai. *Convolutions and Backpropagations*. URL: https://pavisj.medium.com/convolutions-and-backpropagations-46026a8f5d2c. (accessed: 20.11.2022).

[29] Mingxing Tan and Quoc V. Le. “EfficientNetV2: Smaller Models and Faster Training”. In: *CoRR* abs/2104.00298 (2021). arXiv: 2104.00298. URL: https://arxiv.org/abs/2104.00298.

[30] Yaqing Wang et al. *Generalizing from a Few Examples: A Survey on Few-Shot Learning*. 2019. DOI: 10.48550/ARXIV.1904.05046. URL: https://arxiv.org/abs/1904.05046.

[31] Gaowei Xu et al. “A Deep Transfer Convolutional Neural Network Framework for EEG Signal Classification”. In: *IEEE Access* 7 (2019), pp. 112767–112776. DOI: 10.1109/ACCESS.2019.2930958. URL: https://ieeexplore.ieee.org/document/8772136.

[32] R. Yamashita et al. “Convolutional neural networks: an overview and application in radiology”. In: *Insights Imaging* 9 (2018), pp. 611–629. DOI: 10.1007/s13244-018-0639-9. URL: https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9.

[33] Lin YP and Jung TP. “Improving EEG-Based Emotion Classification Using Conditional Transfer Learning”. In: *Frontiers in human neuroscience* 11 (June 2017), p. 334. DOI: 10.3389/fnhum.2017.00334.

[34] Ding-Xuan Zhou. “Universality of Deep Convolutional Neural Networks”. In: *CoRR* abs/1805.10769 (2018). arXiv: 1805.10769. URL: http://arxiv.org/abs/1805.10769.