National Technical University of Athens

School of Electrical and Computer Engineering
Division of Information Transmission Systems and Material Technology

Analysis of EEG recordings with deep learning techniques human-computer interface experiment

Thesis

Angelos Alexopoulos

Supervisor: George Matsopoulos
Professor, National Technical University of Athens

Biomedical Engineering Laboratory
Athens, October 2024

National Technical University of Athens
School of Electrical and Computer Engineering
Division of Information Transmission Systems and Material Technology

Analysis of EEG recordings with deep learning techniques human-computer interface experiment

Thesis

Angelos Alexopoulos

Supervisor: George Matsopoulos
Professor, National Technical University of Athens

Approved by the examination committee on ...th of October 2024

(Signature)	*(Signature)*	*(Signature)*
.....	.....	.....
George Matsopoulos Professor, National Technical University of Athens	Panagiotis Tsanakas Professor, National Technical University of Athens	George Stamou Professor, National Technical University of Athens

Biomedical Engineering Laboratory
Athens, October 2024

National Technical University of Athens
School of Electrical and Computer Engineering
Division of Information Transmission Systems and Material Technology

(Signature)

.....
Angelos Alexopoulos
Electrical and Computer Engineer, N.T.U.A.

Copyright © Angelos Alexopoulos, 2024.
All rights reserved.

The copying, storage and distribution of this thesis, exall or part of it, is prohibited for commercial purposes. Reprinting, storage and distribution for non - profit, educational or of a research nature is allowed, provided that the source is indicated and that this message is retained.

The content of this thesis does not necessarily reflect the views of the Department, the Supervisor, or the committee that approved it.

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Συστηματων Μεταδοσης Πληροφοριας και Τεχνολογιας Υλικων

Ανάλυση εγκεφαλογραφικών καταγραφών με τεχνικές βαθιάς μάθησης πειράματος διεπαφής ανθρώπου-υπολογιστή

Διπλωματικη Εργασια

ΑΓΓΕΛΟΣ ΑΛΕΞΟΠΟΥΛΟΣ

Επιβλέπων: Γεώργιος Ματσόπουλος
Καθηγητής Ε.Μ.Π

Εργαστηριο Βιοϊατρικης Τεχνολογιας

Αθήνα, Οκτώβριος 2024

Abstract

This study introduces a method for analyzing Spectrograms derived from electroencephalogram (EEG) data to classify leg movements. Employing the complex information encoded in EEG signals, we propose a model combining Convolutional Neural Networks (CNNs) and pre-tained models. This approach is designed to extract crucial features from the EEG signals. By applying preprocessing techniques data augmentation, ensemble method etc, the model's ability to capture the patterns associated with different leg movements is greatly enhanced. There are employed different optimizers, such as Adam with different parameters in order to capture the best model's performance. Early results show the model's success in differentiating between specific leg movement tasks, pointing out its potential. This research showcases the importance of signal processing and machine learning in capturing complex brain signals.

Περίληψη

Αυτή η μελέτη εισάγει μια μέθοδο για την ανάλυση των Spectrograms που προέρχονται από δεδομένα ηλεκτροεγκεφαλογραφήματος EEG για την ταξινόμηση των κινήσεων των ποδιών. Χρησιμοποιώντας πληροφορίες που κωδικοποιούνται σε σήματα EEG, προτείνουμε ένα μοντέλο που συνδυάζει Συνελικτικά νευρωνικά δίκτυα (CNN) και προ εκπαιδευμένα μοντέλα. Αυτή η προσέγγιση έχει σχεδιαστεί για να εξάγει κρίσιμα χαρακτηριστικά από τα σήματα EEG. Με την εφαρμογή preprocessig techniques data augmentation, ensemble method etc,το μοντέλο επιδεικνύει ικανότητα αποτύπωσης των μοτίβων που σχετίζονται με διαφορετικές κινήσεις των ποδιών. Χρησιμοποιούνται διαφορετικοί optimizers, όπως ο Adam με διαφορετικές παραμέτρους, προκειμένου να αποτυπωθεί η καλύτερη απόδοση του μοντέλου. Τα πρώτα αποτελέσματα δείχνουν επιτυχία στη διαφοροποίηση μεταξύ της κίνησης ποδιών, επισημαίνοντας τις δυνατότητές του. Αυτή η έρευνα δείχνει τη σημασία της επεξεργασίας σήματος και της μηχανικής μάθησης για τη λήψη πολύπλοκων εγκεφαλικών σημάτων.

Acknowledgements

I would like to thank Mr. Georgios Matsopoulos for supervising this thesis and for giving me this opportunity to work closely in the Biomedical Technology Laboratory thus gaining so much both in knowledge and experience. Also, I warmly thank Mr. Ioannis Kakkos for his guidance and the excellent cooperation we had. Finally, I would like to thank my family and my closest friends.

Contents

Abstract **9**

Περίληψη **10**

Acknowledgements **11**

List of Figures **14**

List of Tables **16**

1 Theoretical Background **17**
1.1 Neurobiology 17
1.1.1 The Human Brain 17
1.1.2 Electroencephalogram(EEG) 19
1.2 Machine Learning 20
1.3 Deep Learning 22
1.3.1 What is Deep Learning 22
1.3.2 How Deep Learning Works 24
1.3.3 CNN Architecture 25
1.3.4 CNN design and blocks 25

2 Previous Work **31**

3 Experiment of the used Data **34**
3.1 Summary 34
3.2 Data Recordings and Participation 34
3.3 Subjects 35
3.4 Experiment setup 35
3.5 Execution 35
3.5.1 Routine of the Experiment 36
3.5.2 Pre-processing of the Data 37

4 Methodology **38**
4.1 Techniques 38
4.1.1 Early stopping 40

4.1.2 Dropout . . . 41
4.1.3 Weight decay . . . 41
4.2 Optimizers . . . 43
4.2.1 Adam . . . 43
4.2.2 RMSprop . . . 43
4.2.3 Gradient Descent . . . 43
4.3 Metrics . . . 45
4.3.1 Recall . . . 45
4.3.2 Accuracy . . . 45
4.3.3 Precision . . . 46
4.3.4 F1-Score . . . 46
4.3.5 Confusion Matrix . . . 47
4.4 ResNet . . . 47
4.4.1 Key Components of ResNet Architecture . . . 48
4.5 EfficientNet . . . 49
4.5.1 Key Components of EfficientNet Architecture . . . 49

5 Data Analysis and Results 51
5.1 Pre-processing . . . 51
5.2 Training Process . . . 53
5.2.1 Model Architectures . . . 56
5.3 Training and Evaluation Process . . . 58
5.4 Results . . . 58
5.4.1 Custom Convolutional Neural Network(CNN) . . . 59
5.4.2 Custom Large Convolutional Neural Network(CNN) . . . 60
5.4.3 Custom Pre-Trained Model(EfficientNet-B0) . . . 62
5.4.4 Custom Pre-Trained Model(ResNet) . . . 64
5.4.5 Overall . . . 66

6 Conclusion &Future Work 74
6.1 Conclusion . . . 74
6.2 Future Work . . . 77

References 79

List of Figures

1.1 Human Brain. . . . 18
1.2 EEG Setup. . . . 19
1.3 Artificial Neuron. . . . 23
1.4 Artificial Neural Network. . . . 24
1.5 A Convolutional Layer. . . . 26
1.6 Pooling types. . . . 27

4.1 Early stopping diagram. . . . 40
4.2 Early stopping parameters. . . . 40
4.3 Dropout diagram. . . . 41
4.4 Gradient Descent Algorithm. . . . 44
4.5 Residual Block. . . . 48
4.6 34 layered Residual Neural Network. . . . 49

5.1 Quick inspection of the Dataframe. . . . 52
5.2 Original vs Transformed image. . . . 53
5.3 Showcase of the 4 different classes. . . . 54
5.4 Script of the dataloader. . . . 55
5.5 Script of the main training of the pipeline. . . . 68
5.6 Script of CNN code. . . . 69
5.7 Script of the Large CNN code. . . . 70
5.8 Confusion Matrix for Custom CNN model. . . . 71
5.9 Confusion Matrix for Custom Large CNN model. . . . 71
5.10 Confusion Matrix for EfficientNet-B0 model. . . . 72
5.11 Accuracy for EfficientNet-B0 model during the training phase. . . . 72
5.12 Graphic representation of Loss for EfficientNet-B0 model during the training phase. . . . 73
5.13 Confusion Matrix for ResNet model. . . . 73

List of Tables

5.1 Summary of SpectrogramClassifier custom CNN Architecture 59
5.2 Experiment Results using ADAM optimizer with the following columns: Accuracy Train(%) Accuracy Test(%), Learning Rate, Weight Decay, Optimizer for the Small CNN. 60
5.3 Experiment Results using SGD optimizer with the following columns: Accuracy Train(%) Accuracy Test(%), Learning Rate, Weight Decay, Optimizer for the Small CNN. 60
5.4 Experiment Results using RMsprop optimizer with the following columns: Accuracy Train(%) Accuracy Test(%), Learning Rate, Weight Decay, Optimizer for the Small CNN. 61
5.5 Summary of SpectrogramClassifier custom Large CNN Architecture . 62
5.6 Experiment Results using ADAM optimizer with the following columns: Accuracy Train(%), Accuracy Test(%), Epoch, Learning Rate, Weight Decay, Optimizer for the Large CNN. 63
5.7 Experiment Results using SGD optimizer with the following columns: Accuracy Train(%), Accuracy Test(%), Epoch, Learning Rate, Weight Decay, Optimizer for the Large CNN. 63
5.8 Experiment Results using RMsprop optimizer with the following columns: Accuracy Train(%), Accuracy Test(%), Epoch, Learning Rate, Weight Decay, Optimizer for the Large CNN. 64
5.9 Summary of EfficientNet-B0 Architecture 64
5.10 Experiment Results using ADAM optimizer with the following columns: Accuracy Train(%), Accuracy Test(%), Epoch, Learning Rate, Weight Decay, Optimizer for the EfficientNet-B0 Architecture and No-Freeze-Backbone. 65
5.11 Experiment Results using SGD optimizer with the following columns: Accuracy Train(%), Accuracy Test(%), Epoch, Learning Rate, Weight Decay, Optimizer for the EfficientNet-B0 Architecture and No-Freeze-Backbone. 65
5.12 Experiment Results using RMsprop optimizer with the following columns: Accuracy Train(%), Accuracy Test(%), Epoch, Learning Rate, Weight Decay, Optimizer for the EfficientNet-B0 Architecture and No-Freeze-Backbone. 66
5.13 Summary of Res-Net Architecture 66

5.14 Experiment Results using ADAM optimizer with the following columns: Accuracy Train(%), Accuracy Test(%), Epoch, Learning Rate, Weight Decay, Optimizer for the ResNet Architecture and employing Freeze-Backbone. 67
5.15 Experiment Results using SGD optimizer with the following columns: Accuracy Train(%), Accuracy Test(%), Epoch, Learning Rate, Weight Decay, Optimizer for the ResNet Architecture and employing Freeze-Backbone. 67
5.16 Experiment Results using RMsprop optimizer with the following columns: Accuracy Train(%), Accuracy Test(%), Epoch, Learning Rate, Weight Decay, Optimizer for the ResNet Architecture and employing Freeze-Backbone. 69
5.17 Best Models of each category 69

Chapter 1

Theoretical Background

1.1 Neurobiology

Neurobiology is the scientific study of the nervous system (the brain, spinal cord, and peripheral nervous system), its functions, and its disorders. Neuroscience investigates the complex workings of neurons, their communication through neurotransmitters, and the role of brain regions such as the hippocampus in memory retention and the prefrontal cortex in making choices. It delves into the processes that allow the brain to change and adapt through neuroplasticity, as a result of learning and experiences. A key aspect of Neurobiology-Neuroscience is the investigation of how we perceive our environment, control our movements, and the neural underpinnings of awareness. Understanding these principles is crucial to deciphering the intricacies of how the brain works and malfunctions, which may lead to new ways to treat neurological conditions.

1.1.1 The Human Brain

The Human brain is the most studied but least understood organ that dictates and direct all the capabilities that a human can exhibit. Abilities such as emotion, touch, memory, vision, breathing, temperature, hunger and all the functions that helps our body stay alive and thrive. It has not revealed all of its secrets yet and it is the most studied field because a wound or malfunction there could alter to the worse the life of a person. The brain and spinal cord that make up the central nervous system(CNS) [1].

The brain transmits and receives electrical, chemical signals inside the body. Various signals dictates various functions, and the brain translates each. There are signals that regulates the tiredness or the pain that a person feels. The thoughts are some of the signals that are kept inside the brain. Though there are other signals are transferred through the spine and across the body's network of nerves. To make that happen, the central nervous system mobilizes the vast amount of neurons (nerve cells) that controls throughtout the body.

At an abstract level, the brain is consisted by the cerebrum, brainstem and

cerebellum.

The cerebrum (front of brain) is made by the gray matter (the cerebral cortex) and white matter at its center. The biggest portion of the human brain, the cerebrum instigate and synchronize the movement and dictates the suitable temperature if the body. There are other parts of the brain and cerebrum taht enable speech, judgment, thinking and reasoning, problem-solving, emotions and learning. And of course there are way more parts of the brain that enables the abilities vision, hearing, touch and other senses.

The cerebral cortex is made up by two halves, or hemispheres. It is sheltered by ridges (gyri) and folds (sulci). The two halves join at a large, deep sulcus (the interhemispheric fissure) that extends from the back of the head to the front of the head. The right hemisphere governs the left side of the body, and the left half controls the right side of the body as easily understood. The two halves coordinates with each other via, C-shaped structure of white matter and nerve routes. These routes are names corpus callosum. In the center of the cerebrum there is the corpus callosum.

The brainstem (middle of brain) attach the spinal cord to the cerebrum with . The brainstem has inside it the midbrain, the pons and the medulla.

The cerebellum has the size of a human like fist of the brain located at the back of the head, under the temporal and occipital lobes and above the brainstem. As the cerebral cortex, it contains two hemispheres. The outer portion contains neurons, and the inner area is in touch with the cerebral cortex. Its function is to govern the voluntary muscle movements and to keep in line the posture, helps with the balance of the body and equilibrium.

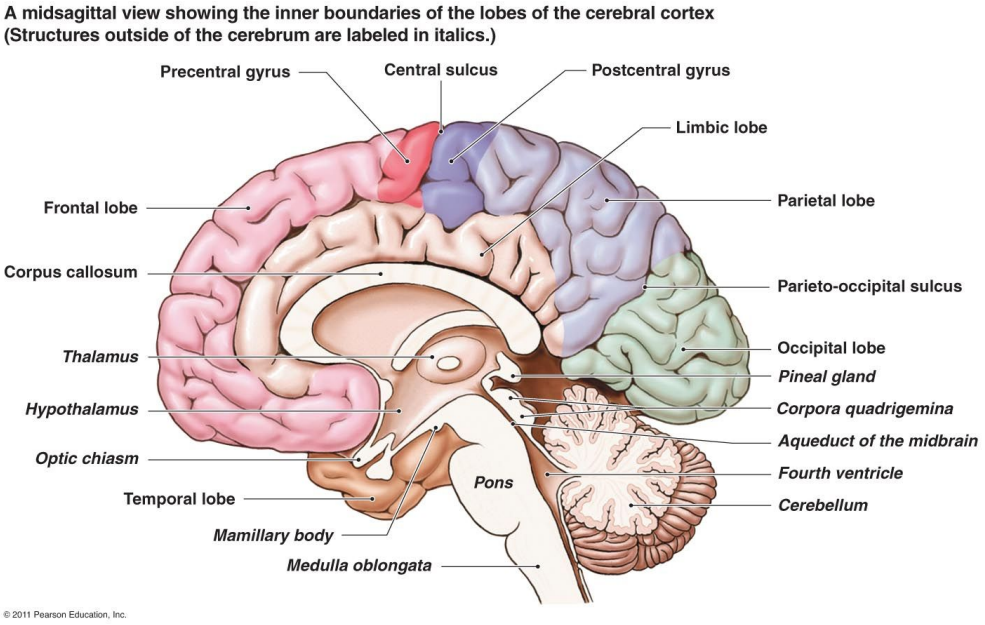


Figure 1.1: *Human Brain.*

1.1.2 Electroencephalogram(EEG)

The data acquired for this thesis was the result of electroencephalogram (EEG) which is a method that documents the electrical activity that is happening in the brain. This recording is happening usually through non-invasive with the use of electrods which are attached to the head. Neurons inside the brain are constantly transmitting and receiving electrical signals. These transmissions are captured by the EEG and could help identify various disorders that are correlated with the brain. Also has been used extensively to recognize epilepsy or other neurological issues.

These alterations in the human brain are detected by the electrods and after that there are amplified by the suitable electronics equipment. After the signal is amplified enough in order to captured by the detectors is recorder in order to be studied. As evident the composition of the human body is different from area to area. This composition affects the resistance of the body and as such the flow of the signal. This has as a result that some neurons do not contribute to these signal recordings. Also some parts of the brain due to their position do not contribute to these recordings Figure 1.2.

The EEG of a normal brain exhibits some specific patterns that depends with the level of mental function.

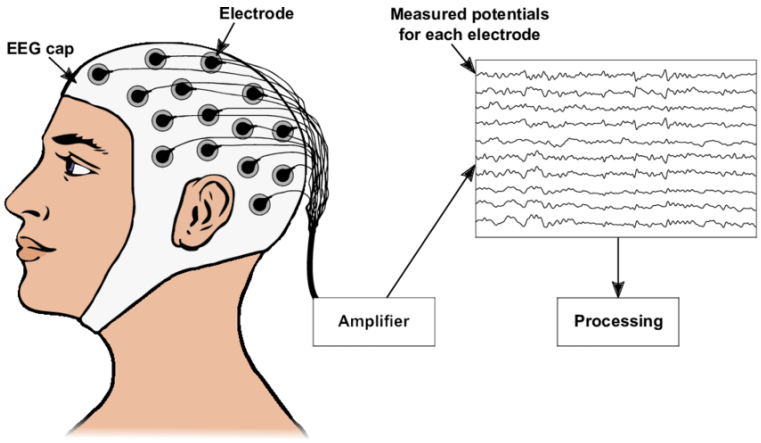


Figure 1.2: *EEG Setup.*

1.2 Machine Learning

The rise of Artificial Intelligence (AI) created a new era of possibilities and directions towards the field of Bioscience. The field of AI started in the mid-20th century and succeeded to reach its current role as a critical part of not only technological advancement bur also part of everyday life of the common people. Early researchers tried the possibility of machines to replicate human cognitive processes.

The investigation of the reasoning can be traced back to the work of philosophers and mathematicians of early antiquity. This investigation of logic was renewed by Alan Turing. His theory was that was possible for a machine to shuffle the 0,1 symbols and recreate any mathematic reasoning. Conjuring discoveries in several other scientific areas such as Information Theory and Neurobiology cleared the path for the capability of an electronic brain.

Initially there were early successes and major breakthroughs. Some of these successes were the creation of a natural language processing programs. Later these decades there were declining funding and interest in these technologies because it failed to catch up with the expectations.

The study of AI was gradually restored in the late 90s by making use of formal methods of mathematics and by succeeding in by coming up with solutions in specific problems. This more narrow focus allowed the researchers to make use of AI in other fields. Deep Learning started to dominate around the 2010s and it was firmly used in many problems. The reason for this domination was mainly based on hardware supremacy and of course the access to huge datasets. Deep Learning renewed the interest and funding of academia and industry to AI. The AI research was skyrocketed by 50% in the years 2015-2019.

Today big industries invest billions of dollars in AI. This technology is used by many industries, driving developments in areas such as automation, entertainment and healthcare. As it advances, the rapid expansion of artificial intelligence is also bringing to light ethical dilemmas about privacy, discrimination and its influence on labor markets, sparking debates about how to steer its trajectory conscientiously. This brief highlights the recurring pattern of AI evolution—from conceptual research to real-world application and societal integration—highlighting the ongoing balance between technology's potential and ethical imperatives.

Machine learning [2], a subset of artificial intelligence, includes a range of computational techniques that enable systems to learn and make predictions or decisions based on data. It includes algorithms that discover patterns and infer information from the data they are exposed to, without being explicitly programmed for specific tasks. This adaptive approach allows machine learning models to improve their performance as they are exposed to more data over time. Central to machine learning is the concept of training, where a model is fed a set of data and learns to recognize patterns associated with performing a specific task, such as classification.

- Initially there is the collection of new data from existing sources. When there are collected, the data go through cleansing, processing, and preparation before analysis can commence.

- After that there is the selection of the appropriate learning machine model for the data analysis. Different models, like decision trees, neural networks, support vector machines, and random forests, possess unique strengths and weaknesses. The chosen model depends on the analyzed data specific application and nature. The use of the Cloud computing supply the resources and abilities for machine learning algorithms to process and investigate large chunks of datasets.

- Closing, the model is trained using the data prepared especially for this model. This assumes feeding the model a very big dataset and allowing it to learn without intervention.

1.3 Deep Learning

Deep Learning is a division of ML methods and functions that utilize the theory around neural networks. This specific area of ML is inspired by the the theory of Biology around artificial neurons. The logic behind this, is that grouping artificial neurons into layers and training them can lead them to create decisions. The word deep is referring to the fact that utilize multiple layers.

1.3.1 What is Deep Learning

The main difference among deep learning and traditional machine learning is the formation of the inside neural network structure. Machine learning models that are not "Deep",used traditionally make use of simple neural networks with one or two computational layers. Deep learning models make use of typically hundreds or thousands of layers to train the models.

While supervised learning models request structured and labeled input signals(data) to produce accurate outputs, deep learning models can make use also of unsupervised learning. These deep learning models can extract some crucial characteristics utilizing these functions of un-supervised functions. Another thing that is extracted are features and connections.

An artificial neuron is a connection point in an artificial neural network. It is the try of mathematics to recreate a biological neuron using functions. This mathematics model could simulate the brain activity to conceive the outside world and interact with it, receiving and transmitting signals and decisions[3].

As illustrated in Figure 1.3, an artificial neuron consists of several components: inputs, relative weights, bias, activation function, and finally, output. Inputs are the data received by the neuron, depending on the dendrites of a biological neuron. Weights are assigned to each input to shape its importance or impact. The bias acts as a threshold that the neuron must cross to become active.

Inside an artificial neuron, the main function is to calculate the sum of the inputs multiplied by their respective weights and then add the bias. This set is processed through an activation function, which is responsible for producing the output of the neuron. The activation function introduces non-linearity, which is crucial to the neural network's ability to capture and represent complex data patterns.

Structure:

- **Neurons:** The core of an Artificial Neural Network. This part is conceived in such way to recreate the function of biological neurons in a human body. This neurons like blocks takes in input, they use this input along with hard wired information and eventually produces an output based on the input received.

- **Weights:** Weights are the variables inside every neuron-layer that along with the received input data define the output. These variables are changed accordingly during the training process with the goal of transforming the actual output as much as possible towards the target output.

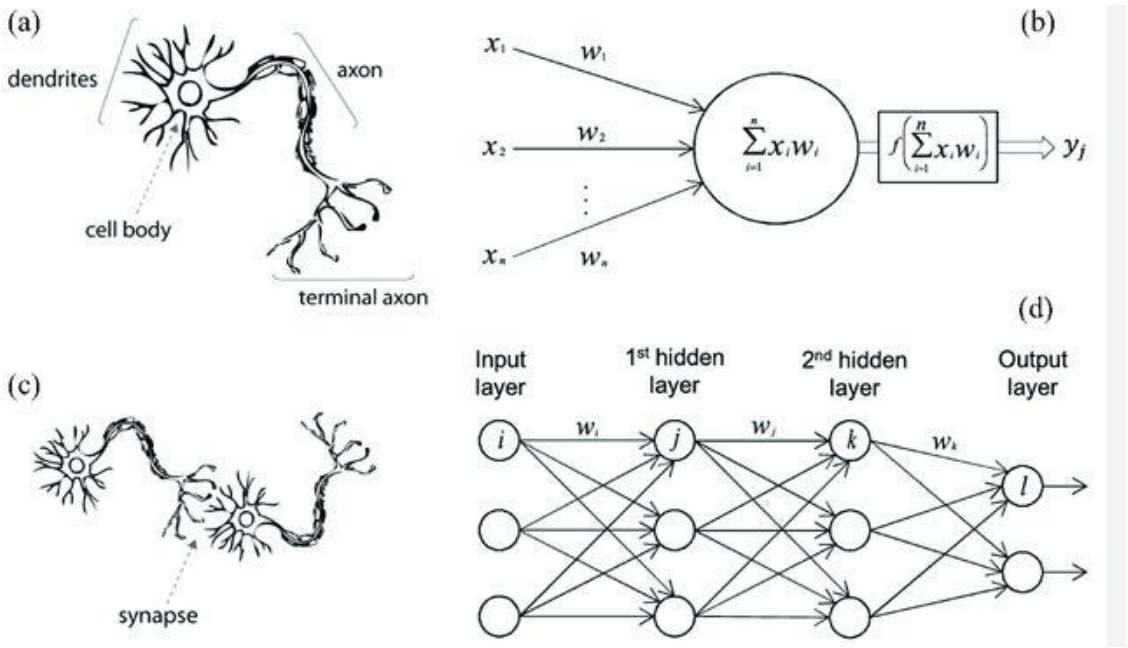


Figure 1.3: *Artificial Neuron.*

- **Bias:** The bias parameter is present in order to direct the output fits the input. This parameter is tuned with the weights with the help of ML algorithms. The Bias is different from the weights because Bias is not dependant to the layers before. It depends on the model but usually the Bias is initialised in a value of 0.

- **Layers:** Artificial Neural Network is made up from many layers. The information flows through these layers and every output of a previous layer is the input of the next one. This sequential process continues until the output of the layer is the the final output of the classification. Each layer has its own purpose in the network and perform some pre defined calculations. These calculations introduce the absence of linearity needed in a neural network. These layers are the input layer, one or more hidden layers, and an output layer. The input layer receives the initial data for processing, the hidden layers perform computations with weights and biases, and the output layer produces the final prediction or classification.

- **Activation Function:** This is a mathematical function which is applied to the input sum in the output of a neuron and enables the function to learn recognize patterns and create them. Also this function is applied to the weights and bias, and as a result determine the neuron's output. Most used activation functions are the sigmoid, tanh, ReLU (Rectified Linear Unit), and softmax.

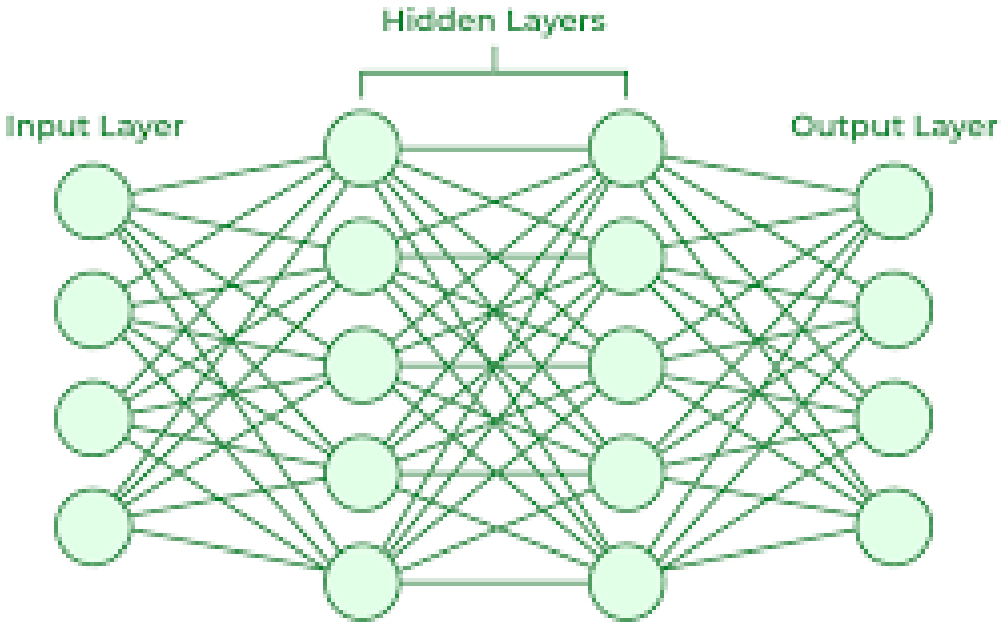


Figure 1.4: *Artificial Neural Network.*

1.3.2 How Deep Learning Works

Neural networks, or artificial neural networks, tries to recreate the brain of the humans throughout a blend of data inputs, weights and bias—all acting as silicon neurons. These elements are working together to recognize, categorize and find patterns inside the data.

Deep neural networks are made of multiple layers of nodes that are connected between them, each making up on the previous layer to purify and optimize the categorization. This progression of computations through the network is called forward propagation. The input and output layers of a deep neural network are called visible layers. The input layer is where the deep learning model ingests the data for processing, and the output layer is where the final prediction or classification is made.

Another process called back-propagation employs algorithms, such as gradient descent, to calculate errors in predictions, and then adjusts the weights and biases of the function by moving backwards through the layers to train the model. Together, forward propagation and backpropagation enable a neural network to make predictions and correct for any errors .Over time, the algorithm becomes gradually more accurate.

Deep learning requires a huge amount of computing power. High-performance graphical processing units are perfect because they can handle a large volume of calculations in multiple cores with copious memory available. Distributed cloud computing might also assist. This level of computing power is necessary to train deep

algorithms through deep learning. However, managing multiple GPUs on premises can create a large demand on internal resources and be incredibly costly to scale. For software requirements, most deep learning apps are coded with one of these three learning frameworks: PyTorch or TensorFlow.

1.3.3 CNN Architecture

Convolutional Neural Networks (CNNs) represent a category within deep neural networks. They excel at handling data that exhibits a network like structure, with photos being the best example. What sets CNNs apart is their special architecture, which is engineered to learn hierarchical spatial features from input data in an autonomous and flexible manner. This intrinsic capability makes CNNs particularly good at solving various computer vision challenges. Their applications can vary to a big extent, solving issues such as analyzing and recognizing images and videos, categorizing visual content, and identifying objects within images [4].

The power of Convolutional Neural Networks lies in their capacity to extract and learn relevant features from visual data, making them a basic technology in the field of computer vision and image processing.

1.3.4 CNN design and blocks

Convolutional layer

The convolutional layer Figure 1.5 is the key core component of a CNN(convolutional neural network), and it is where the plethora of calculations occurs. It needs a few components, which are the input data, a filter, and a feature map. Let's accept that the input will be a color image, which is made up of a matrix of pixels in 3D. This means that the input will have three dimensions—a height, width, and depth—which correspond to RGB in an image. We also have a feature detector, also known as a kernel or a filter, which will move across the receptive fields of the image, checking if the feature is present. This process is known as a convolution [5].

The feature detector is a two-dimensional (2-D) array of weights, which represents part of the image. While they can vary in size, the filter size is typically a 3x3 matrix; this also determines the shape of the receptive field. The filter is then applied to an area of the image, and a dot product is calculated between the input pixels and the filter. This dot product is then fed into an output array. After that, the filter shifts by a stride, repeating the process until the kernel has swept across the entire image. The final output from the series of dot products from the input and the filter is known as a feature map, activation map.

Note that the weights in the feature detector remain fixed as it moves across the image, which is also known as parameter sharing. Some parameters, like the weight values, adjust during training through the process of back propagation and gradient descent. However, there are three hyper parameters which affect the volume size of the output that need to be set before the training of the neural network begins. These include:

- The number of filters influence the depth of the output. For instance, three distinct filters would give three different feature maps, creating a depth of three.

- Stride is the distance, or number of pixels, that the kernel moves over the input matrix. While stride values of two or greater is rare, a larger stride yields a smaller output.

- Zero-padding is mostly used when the filters do not fit the input image. This sets all elements that fall outside of the input matrix to zero, producing a larger or equally sized output. There are three types of padding Valid padding, Same padding and Full padding.

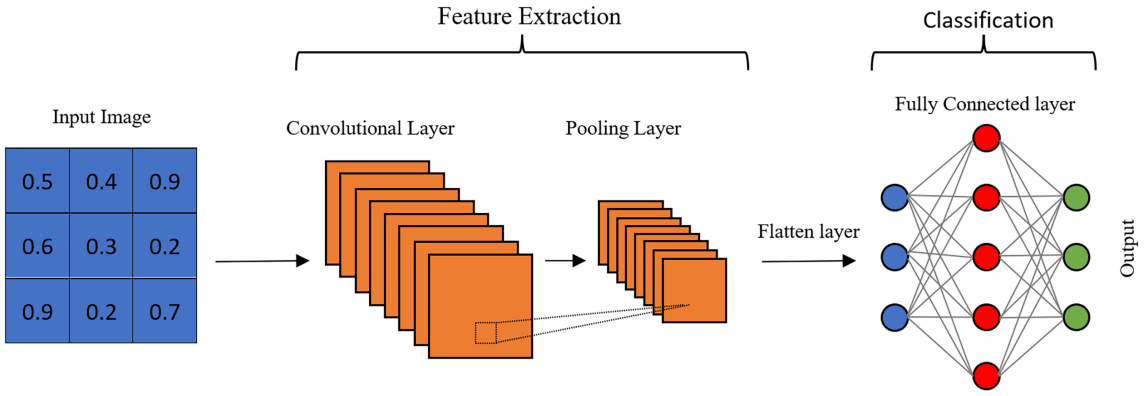


Figure 1.5: *A Convolutional Layer.*

Pooling Layers

Pooling layers, which are known also as down-sampling, reduce the dimensions, reducing the volume of the parameters in the input. Just like the convolutional layer, the pooling operation scans a filter across the entire input signal, but the difference is that this filter bear no weights. On the contrary, the kernel applies an aggregation function to the values within the receptive field, populating the output array. There are two main types of pooling:

- **Max pooling**: As the filter moves across the input, it selects the pixel with the greatest value to give to the output array. As an aside, this approach tends to be used more often compared to average pooling.

- **Average pooling**: As the filter moves towards the input, it measures the average value within the receptive field to send to the output array.

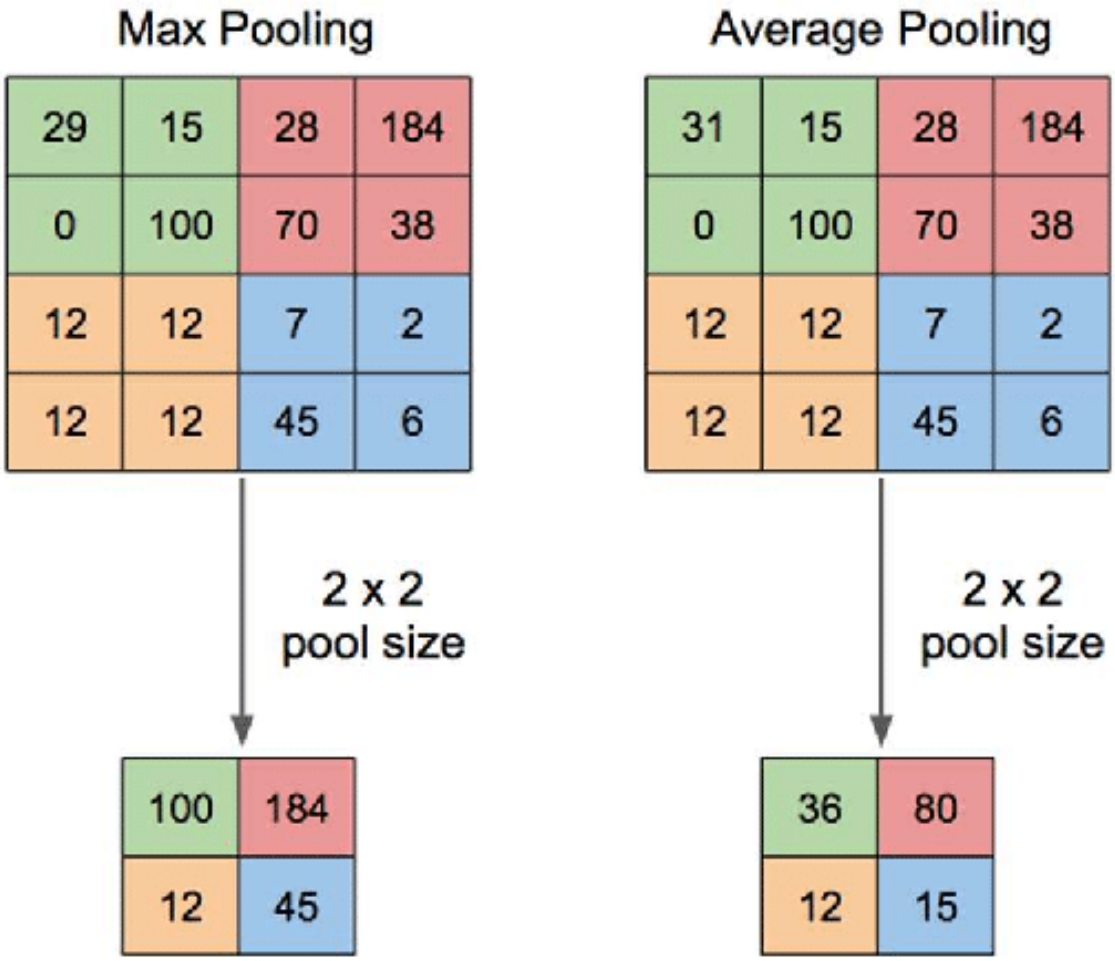


Figure 1.6: *Pooling types.*

Activation Functions

In the course of the training, an activation function is applied to the model to introduce non-linearity, and allowing it to recognize more complex patterns. These functions are cornerstone for matching inputs to outputs. Essentially this function chooses what neurons are going to be activated and what not. This decision is made upon the inputs that is fed in the model. These functions calculate the weighted sum of the combined biases and input signals. The result of this functions is the output that decides the neuron's activation. In CNNs, there are non-linear activation functions like fully connected (FC) and convolutional layers. These functions make the neural network able in capturing complex patterns and recognize them. The most popular activation functions are:

- **ReLU**: Mostly used in CNNs, it transforms all the negative inputs to zeros. This function simplifies the calculations but sometimes can cause the neurons to stop functioning. This activation function helps create a non-linear behavior that enables the network to learn complex patterns, is efficient in terms of

calculations and helps avoid the problem of of the vanishing gradient.

$$ReLU(x) = \max(0, x) \tag{1}$$

The above equation means that given any input **x**, the result of this function returns **x** if the input **x** is positive or **0** if the input **x** is negative.

- **Sigmoid**: Convert the analog inputs into a range from 0 to 1 using an S-like curve. The output is always between 0 and 1. If the input **x** is a large positive number then the output approaches 1. If on the contrary is a large negative number then it approaches 0. And when the input is 0 then the result equals 0.5.

$$sigma(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

where:

  - **e**: the mathematical constant (approximately 2.71828).
  - **x**: the input value.
  - e^{-x}: the exponential function with negative **x** as power.

- **Tanh**: Just like the sigmoid function. On this case the output values varies between -1 to 1, making use of a zero-centered range. The output is in between -1 and 1 which helps because is centered around 0. If the input **x** is a large positive number then the output approaches 1. If on the contrary is a large negative number then it approaches -1. The zero-centered feature helps because it makes training of neural networks easier since the outputs of the network are both positive or negative.

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3}$$

where:

  - **e**: the mathematical constant (approximately 2.71828).
  - **x**: the input value.
  - e^{-x}: the exponential function with negative **x** as power.
  - e^{x}: the exponential function with positive **x** as power.

Fully-connected layer

In a fully-connected layer the pixel values of the input image are not interconnected to the output layer in partially connected layers. However each node in the output layer connects directly to a node in the previous layer.

This layer performs the task of classification based on the features extracted through the previous layers and their different filters. While convolutional and pooling layers tend to use the ReLu functions, FC layers usually leverage a softmax activation function to classify inputs suitable, producing a probability from 0 to 1.

Softmax Layer

In the majority neural network designs, especially those used for classification tasks, the softmax layer plays a crucial role at the stage of the output. This layer calculates the generic scores from the previous layer, transforming them into probabilities through the application of the softmax function.

The softmax function is specially designed in a way to produce output values in the range of (0, 1), with the added requirement that these values have a sum of 1. This feature allows the outputs to be interpreted as probabilities, which is particularly useful in classification scenarios.

When applied to the logits from the final layer of a neural network, the softmax layer effectively transform these raw scores into a probability distribution across the possible classes. The mathematical definition of the softmax function for a given score vector z forms the basis of this transformation. This approach enables neural networks to provide not just classifications, but also the confidence levels associated with each class prediction, enhancing the interpretability and usefulness of the model's output. The softmax function for a score vector z is defined as:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{4}$$

where:

- **e**: the mathematical constant (approximately 2.71828).
- z_i: the input value.
- e^{z_i} : the exponential of the input.
- $\sum_j e^{z_j}$: sum of exponentials of all classes.

A key attribute of the softmax layer is its ability to constrain the output values between 0 and 1 while ensuring that they sum to 1. This feature produces a valid probability distribution, making the softmax function particularly valuable in multi-class classification scenarios. It allows a clear probabilistic interpretation of how the model assigns probabilities to various classes.

The softmax function transforms the raw scores into a probability distribution, when applied to the final layer outputs of a neural network. This permits users to interpret the highest likelihood as the model's predicted class for a given input.

The softmax layer serves as a bridge between a model's internal computations and human interpretable probabilities, enhancing the usability and interpretability of the model in real-world applications. It provides a standardized way of expressing a model's confidence in multiple possible outcomes, which is critical in many machine learning tasks.

Loss Functions

A loss function is employed to measure model performance by calculating the deviation of a model's predictions from the correct, "ground truth" predictions. This discrepancy between predicted and actual outcomes finally defines the output of the classification. The loss function takes in account two parameters: the predicted output (or prediction) and the actual output (or label). As evident these functions make use of the theory of mathematics and as such there are different loss functions. The are many suitable mathematic equations that are used as Loss Functions in the Machine Learning field. The use of each function is depending to the problem. Next it will be presented the Loss Function used in this particular task:

- **Cross-Entropy Loss Function**: This is most common found in classification problems. The logic behind it is, that measures the performance of the CNN model by converting the probability between 0 and 1. In multi class classification, just like in this thesis, it works with the softmax to create a propability distribution.

$$p_i = \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}} \tag{5}$$

  where:

  - p_i**:** this is the probability of the i class.
  - e**:** Euler's number.
  - a_i**:** Input value for the class i.
  - N**:** Total number of the classes.
  - $\sum_{k=1}^{N}$**:** Sum of all classes for the total of the classes.

  With this equation, the model outputs are converted into probabilities, the sum of all probabilities are added to 1 and there is comparison based on the probabilities thresholds.

Chapter 2

Previous Work

This thesis aims to classify the movement of a limb using EEG signals using Deep Learning. After gaining spectograms of an EEG signal through an experiment a neural network is developed in order to achieve the main goal. Several research articles have utilized this model in similar ways. Following there is a summarizing of several researches in similar direction to the aim of this thesis.

In this research paper [6] is presented the accurate classification of upper limb movement using electroencephalogram signals. Correct classification of the movement of upper limbs could be of great help patients who lack this ability. With the help of a Brain-Computer-Interface(BCI) there are executed some actual movement and is tried to be distinguished if there were real or imaginary. The shown study shows the ability of identifying and classifying activities related to movement of the upper limbs(actual or imagined) by utilizing an EEG-based brain-computer interface. In order for this to be achieved, a public EEG dataset was employed. This dataset contained recorded EEG signals of fifteen different subjects measured with a 61-channel EEG device. These four different categories of movements were classified for each of the subjects based on the spectrograms generated from the EEG signals and utilizing pre-trained Deep Learning models. This idea was met with success as the results showed how successful was. The mean classification accuracy of the four movements categories reached 87.36%, using the proposed from the paper method. Also one of the participants achieved 97.03%. This study concluded, using as datasets of clinical data, that utilizing spectograms of EEG signals and Deep Learning techniques can classify with high accuracy the upper limb movement.

In the shown research [7] is studied and presented the movement of the right upper limb which is something different than the most studies which focus on both limbs. The proposed methodology includes a multi-branch fusion convolutional neural network for adapting to the features of the used raw EEG signals. Also there are utilized the 2-D time frequency maps for the representations of the features and the signal recognition of the right upper limb. More specifically three task were tested, arm extension(binary classification), wrist rotation and object grasping(binary classification) on 25 participants. For the binary classification between the grasping object and arm-extending the results were really good achieving 78.52% of accuracy

and 0.57 of kappa value. When all three task were taken into account the results were 57.06% of accuracy and 0.36 of kappa value. These results showcased that this MF-CNN(multi-branch convolutional neural network) achieves better results over conventional methods and algorithms. Concluding, this MF-CNN proposed in this study, is using both the time-domain and frequency-domain of the features of EEG signals. These features show great results in classifying correctly the single arm task and have much potential for further studies.

Another relevant study, [8] propose an innovative approach to EEG-based motor imagery classification using a multi-layer CNN architecture with feature fusion. Their method involves extracting features from different layers of the CNN and combining them to create a more comprehensive representation of the EEG signals. This feature fusion technique has the goal of capturing both low-level and high-level features of the motor imagery tasks. The evaluation of their approach was done on various motor imagery datasets. These datasets includes lower limb movements. Their results showed improved classification accuracy compared to single-layer CNN models and traditional machine learning methods.

In a similar direction this research is working towards the challenge of detecting the onset of lower-limb movements using EEG signals. In the [9] the researchers developed a system that performs both continuous classification and asynchronous detection of movement onset. They used a combination of common spatial pattern (CSP) for feature extraction and support vector machines (SVM) for the classification part of the project. Another thing is that they did not use the traditional CNNs and their approach to continuous and asynchronous detection could be valuable for more research to this topic. The paper also provides insights into the specific challenges of lower-limb movement detection from EEG data.

Another idea proposed in this [10] is the idea of using a model which is a combination of long short-term memory (LSTM) and stacked autoencoder (SAE). The key for higher classification accuracy in EEG data analysis lies in the effective representation of the features of EEG. This approach focus on one single unique frequency before the application of any learning model. However, this could limit its performance since different information is usually carried in every frequency band of EEG signals. The proposed system mainly investigates a fully channel deep learning model that learns features from different frequencies to accurately classify various upper limb movement intentions.

For the validation of the proposed method, four high-level amputees performed five different motor intention tasks. The recorded EEG signals were pre-processed and further explored the influence of input representation on the performance of LSTM-SAE by adding four frequency bands related to tasks. Then, t-distributed stochastic neighbor embedding was applied to reduce the redundancy of features and further improve the learning ability for the motor intention recognition task.

The results achieved in the previous presented research were very promising, since the overall average performance of the proposed model showed an accuracy of 99.01% with a precision of 99.10%, recall of 99.09%, F1 score of 99.09%, and specificity of 99.77%, with Cohen's kappa at 99.0% across subjects and classes.

This is further explained by the use 2-dimensional t-SNE, based on the fact that multi-class separation in feature space is not distinct; it clearly appeared in the case of the performed signal decomposition.

Chapter 3

Experiment of the used Data

3.1 Summary

The data used in this very thesis were acquired in the context of the project "IMPLEMENTATION OF AN INTEGRATED SUPPORT AND REHABILITATION OF MOTOR HYBRID BRAIN-COMPUTER INTERFACE" by the UNIVERSITY OF WEST ATTICA. This project investigate into motor mechanisms and utilizes multi-channel recordings from Electro-encephalography (EEG), Electro-oculography (EOG), and Electro-myography (EMG) during the planning and execution of both actual and imagined movements (MI). The initial phase involved analysis of brain activity and search of functional connections through the brain using graph theory techniques. In addition this study investigated the information flow and responses of intermediate muscles during the transmission of brain commands to the muscles. For the successful integration of these diverse recordings, it was necessary the implementation of parallel and separate processing methods for each signal type. Subsequently, a combined analysis of EEG and EMG recordings was conducted to correlate brain activity with kinetic degrees of freedom.

3.2 Data Recordings and Participation

This research experiment includes short duration electro-encephalographic and electro-myographic signal recordings by employing non-invasive Electroencephalography,Electro-oculography and Electro-myography, while planning and executing simple movements(like walking). These recordings took place in NTUA(National Technical University of Athens) using as subjects, healthy volunteers. All participants where required to give written consent, whereas confidentiality of the research records is guaranteed by anonymization.

3.3 Subjects

The subjects of this experiment were 30 people, both male and females, and were pre-screened in order for the assurance of their eligibility in participation. Reasons of disqualification are: history of psychiatric, neurological or cognitive disorders; prescription of long-term medication; sleep disorder or childhood history of ADHD; lack of full night sleep (more than 7 hours) for two nights prior the experiment; history of any hearing problem. On the experiment day, participants were requested to restrain from caffeine and alcohol consumption 2 hours prior the experiment.

3.4 Experiment setup

This experiment is conducted inside a Faraday room in order to monitor every electromagnetic interference that could interfere with measurements. Also every participant was galvanic isolated using optical receiver for trigger inputs. The electrodes were bundled in order to reduce magnetic interference.

The equipment used were multi-channel EEG/EOG/EMG devices using a 64-channel electrode cap (Biosemi, Active two System) according to the standard 10–20 system with 8 additional channels for EOG and EMG recordings.

3.5 Execution

Initially participants were asked to fill in the age, field of study, dominant hand and driving experience in years. After that they were given questionnaire to assess the multiple dimensions of subjective state response in experimental environments, referring to the energetic arousal, motivation, and concentration needed to accomplish the task.

All signals were digitized with a sampling rate of 512 Hz. After that Anti-aliasing method is performed with a band-pass filter($0.5 - 70$ Hz), and in order to avoid main interferences a 50 Hz notch filter was used. The electro-physiological recordings:

- **EEG:**Used 64 channels with electrode positions : Iz, Fp1, AF7, AF3, F1, F3, F5, F7, FT7, FC5, FC3, FC1, C1, C3, C5, T7, TP7, CP5, CP3, CP1, P1, P3, P5, P7, P9, PO7, PO3, O1, Oz, POz, Pz, CPz, Fpz, Fp2, AF8, AF4, AFz, Fz, F2, F4, F6, F8, FT8, FC6, FC4, FC2, FCz, Cz, C2, C4, C6, T8, TP8, CP6, CP4, CP2, P2, P4, P6, P8, P10, PO8, PO4 and O2.
- **EOG:**Used 2 channels bipolar electrodes placed at the outer canthi(HEOG) of the eyes and above and below(VEOG) the right eye
- **EMG:**Used 4 channels 2 on each leg in Tinialis anterior, Gastrocnemius positions.

3.5.1 Routine of the Experiment

Each and everyone of the participants undertook a 4-session trial consisted of two real motion and two imaginary motion movements. Each task was consisted of 20 trials (4 conditions balanced based on the stimuli presented) and had a duration of 6.5 minutes. The visual stimuli were presented for about 3.5 seconds to the participants while before each stimulus a fixation cross is shown for 1.5 sec. Each stimulus showed an image of two footsteps in the center of the screen in black background, where right, left, right and left or no footsteps are highlighted, resulting in four indication conditions. Throughout the task, a fixation cross remains visible. Acoustic cues are presented at 5-second intervals, with each individual cue having a duration of 1 second. The tasks was done with the framework: Psychopy utilized in Python. More over the four tasks include:

- **Visual Real Motion:** where the stimuli conditions were four, right, left, both or none and for each case the participant were required to move the corresponding leg in respond to optical highlight of right, left, both or none footstep.
- **Visual Imaginary Motion:** where stimuli conditions were the identical to Visual Real Motion, but participants had to imagine performing the motion requested and not actually moving.
- **Auditory Real Motion:** where the stimuli conditions were four, right, left, both or none and for each case the participant were required to move the corresponding leg in respond to acoustic signal of right, left, both or none footstep.
- **Auditory Imaginary Motion:** where stimuli conditions were the same as Auditory Real Motion, however participants had to imagine performing the motion requested and not actually moving.

3.5.2 Pre-processing of the Data

For the needs of the experiment there was established a standard pre-processing pipeline. This was consisted of FIR band-pass filters with frequencies ranging from 1 to 40 Hz. This meant that the Finite Impulse Response band-pass filter kept only those frequencies. After that these signals were re-referenced using as common reference the average signals from all signals. Also the ocular artifacts were removed using ICA(Independent Component analysis) which removed signal components that had correlation with the eye movement.

After these steps, signal trends were removed from the EEG signals and were re arranged into smaller parts(epochs). Each of this epoch was synchronized 100 ms before the stimuli was shown. After these steps only the correct target responses were qualified for further study.

Chapter 4

Methodology

4.1 Techniques

Regularization is a group of methods for lowering over-fitting in machine learning models. Usually, regularization trades a marginal decrease in the accuracy of the training set for an increase in transferability. Overfitting happens when a model learns the training data too good, taking in account also noise-bias along with the underlying patterns, which can negatively affect its performance on new data. Regularization encompasses a range of techniques to correct for overfitting in machine learning models. Like the previous, regularization is a method for increasing a model's transferability, that is, its ability to produce accurate predictions on new datasets. Regularization gives this bigger transferability sacrificing the lesser training error. Paraphrasing, regularization methods usually lead to worse accurate predictions on training data but more accurate predictions on testing data.

This compromise of increased training error for less testing error is known as bias-variance trade-off. Bias-variance tradeoff is a well-known issue in machine learning. It's necessary to first describe "bias" and "variance." To refer to it shortly:

- **Bias:** calculates the mean difference between predicted values and true values. As bias goes up, a model forecast less accurately on a training dataset. High bias refers to high error in training.
- **Variance:** calculates the difference between predictions across various realizations of a given model. As variance increases, a model forecast less accurately on unseen data. High variance refers to high error during testing and validation.

Bias and variance thus inversely represent model accuracy on training and test sets respectively. On the same time reduction in both is not always possible, resulting in the need for regularization. Regularization decreases model variance at the cost of increased bias.

By increasing bias and decreasing variance, regularization can solve the problem of overfitting. Overfitting occurs when error on training data decreases but on the

same time error on the testing data stops of decreasing or begins increasing. Paraphrasing, overfitting describes models with low bias and high variance. However, if regularization introduces too much bias, then a model will underfit. The regularization technique used in this particular project is the L2 (Ridge) regularization also called **Weight Decay**. For this parameter **Weight Decay** there were tested various values to get the parameter that gave the best result. This technique add some offset into the Loss Function that "punish" the the model's weights squared magnitude if it gets too big.

4.1.1 Early stopping

This procedure tries to stop the process of training before the model starts learning unnecessary information like the noise within the model. This procedure risks stopping the training process too early, leading to the opposite issue of underfitting. Finding the "sweet spot" between under fitting and over fitting is the main goal here Figure 4.1.

The **Early Stopping** technique was initialized in this project by setting the numbers of epochs that the model would investigate if no improvement is achieved. In our case this parameter was set to **5 epochs** Figure 4.2. If this condition is met then the model quits the training because there is no visible improvement in terms of metrics.

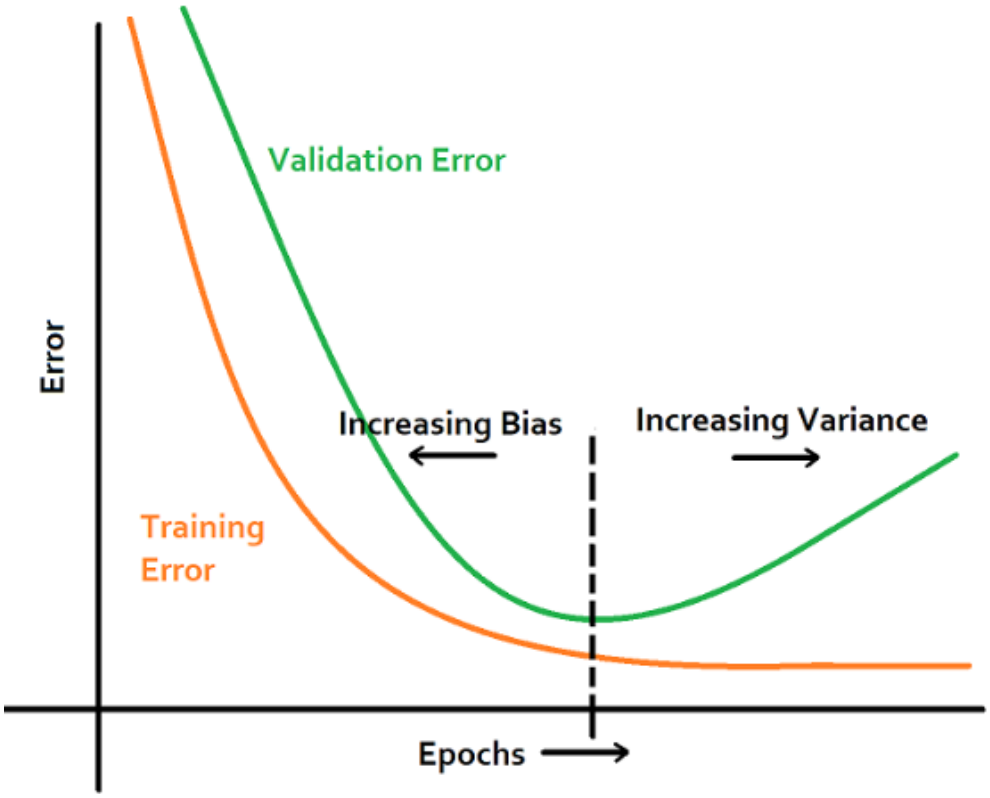


Figure 4.1: *Early stopping diagram.*

```
5  device = 'auto' # 'auto' or 'cpu' or 'cuda:0'
6  epochs = 300    # epoch means all data has been passed through the network once
7  delta = 1e-6    # for early stopping. If the difference between the current loss and the previous loss is less than delta, stop training
8  early_stopping = 5  # no improvement, epochs to exit
```

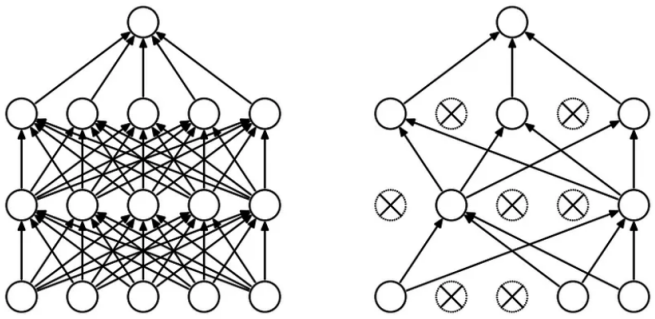
Figure 4.2: *Early stopping parameters.*

4.1.2 Dropout

Dropout is another regularization technique for neural networks that drops a unit (along with all of its connections) at training time with a certain probability to prevent the neurons from learning the specific dataset and thus avoiding overfitting.

The idea is to prevent co-adaptation, where the neural network becomes too reliant on particular connections, as this could be symptomatic of overfitting. Intuitively, dropout can be thought of as creating an implicit ensemble of neural networks . The resultant network simulates training diverse architectures, leading to a model less prone to overfitting and with improved robustness to varied input data Figure 4.3. The Dropout is usually need in the latest layers as they tend to overfit more. Also in the early layers most of the training is happening so is sould not be dropped too much. Also the dropout should be applied in after the activation functions because it helps avoid the co adaptation of the neurons.

The rate of **Dropout** is defined by a percentage of neurons to be dropped. In the models used in this project dropout was implemented in the **Custom Large CNN** as shown in the Figure 5.7. The Dropout was added two times in the code and its parameters were 25% and 50% percent respectively. For the **Small CNN** there was no need for Dropout as its size did not justify it. As bigger models need generally regularization the pre trained **EfficientNet** used also Dropout. The ratio in this model increased linearly from 20% to 50%. On the contrary the **ResNet** did not employ any form of Dropout.



(a) Standard Neural Net (b) After applying dropout.

Figure 4.3: *Dropout diagram.*

4.1.3 Weight decay

Weight decay, also known as L2 regularization, is a fundamental technique used in deep learning to improve model performance. It acts as a regularizer that penalizes large weights in the network, leading to several benefits [11]:

- **Reduces Overfitting:**Large weights can lead to the model memorizing the training data and failing to generalize to unseen examples. Weight decay

penalizes large weights, encouraging the model to learn smaller weights that capture the underlying patterns in the data rather than memorizing specific details. This leads to better generalization performance on unseen data.

- **Improves Model Stability:**Large weights can make the training process unstable and sensitive to noise in the data. Weight decay helps to stabilize the training process by preventing the weights from becoming too large. This makes the model less prone to overfitting and improves its overall robustness.

- **Promotes Feature Sharing:** Weight decay encourages the model to learn weights that are similar across different neurons. This promotes feature sharing, where a single feature is used by multiple neurons in the network. This can lead to a more efficient model with fewer parameters.

- **Improves Generalization in Overparameterized Models:**In modern deep learning, models often have many more parameters than the amount of data they are trained on. This phenomenon is known as overparameterization. Weight decay helps to control the complexity of overparameterized models and improves their generalization performance.

How it Works:

Weight decay adds a penalty term to the loss function that is proportional to the sum of the squared weights in the model. This penalty term encourages the model to learn smaller weights during training.

The weight decay penalty is typically implemented in one of two ways:

- **L2 regularization:** This directly adds a term to the loss function that is proportional to the sum of the squared weights.

- **Choosing the Right Weight Decay:** The optimal value of the weight decay parameter depends on various factors such as the size and complexity of the model, the amount of training data, and the learning rate. It is important to find the right balance between preventing overfitting and optimizing model performance. Techniques like grid search, hyperparameter optimization, and cross-validation can be used to find the best value for weight decay.

In this specific case the suitable parameters of **Weight Decay** were investigated during the training from a pool of values (0, 0.1,0.01, 0.001, 0.0001).

4.2 Optimizers

Optimizers are techniques or algorithms used to decrease the value of loss (an error) by tuning various parameters and weights, hence minimizing the loss function, making the accuracy better and train the the model faster [12]. In this particular thesis the following three optimizers **Adam**, **Gradient Descent** and **RMSprop** were used and tested.

4.2.1 Adam

Adaptive Moment Estimation is an algorithm for optimization technique for gradient descent. The method is really efficient when working with large problem involving a lot of data or parameters. It requires less memory and is efficient. Intuitively, it is a combination of the 'gradient descent with momentum' algorithm and the 'RMSP' algorithm. Adam optimizer involves a combination of two gradient descent methodologies [13].

4.2.2 RMSprop

RMSProp (Root Mean Squared Propagation) is an adaptive learning rate optimization algorithm. It is an extension of the popular Adaptive Gradient Algorithm and is designed to dramatically reduce the amount of computational effort used in training neural networks. This algorithm works by exponentially decaying the learning rate every time the squared gradient is less than a certain threshold. This helps reduce the learning rate more quickly when the gradients become small. In this way, RMSProp is able to smoothly adjust the learning rate for each of the parameters in the network, providing a better performance than regular Gradient Descent alone. The RMSprop algorithm utilizes exponentially weighted moving averages of squared gradients to update the parameters.

4.2.3 Gradient Descent

Gradient descent is an optimization algorithm which is commonly-used to train machine learning models and neural networks. It trains machine learning models by minimizing errors between predicted and actual results.

Training data helps these models learn over time, and the cost function within gradient descent specifically acts as a barometer, gauging its accuracy with each iteration of parameter updates. Until the function is close to or equal to zero, the model will continue to adjust its parameters to yield the smallest possible error.

The cost function quantifies the difference between the model's predictions and the actual data. The aim of gradient descent is to identify the parameter values that minimize this error, thereby enhancing the model's predictive accuracy.

Gradient descent, in its practical application, assesses the cost function's steepness at a specific point and moves in the opposite direction—towards the gradient's

negative. This approach is taken because moving counter to the gradient results in a reduction of the cost function's value. The learning rate, a key hyper parameter, determines the magnitude of steps taken in each gradient descent iteration. This parameter is critical as it influences the speed at which the algorithm reaches a minimum. If the learning rate is set too high, it may cause the algorithm to overshoot its target, while a rate that's too low can result in slow convergence Figure 4.4.

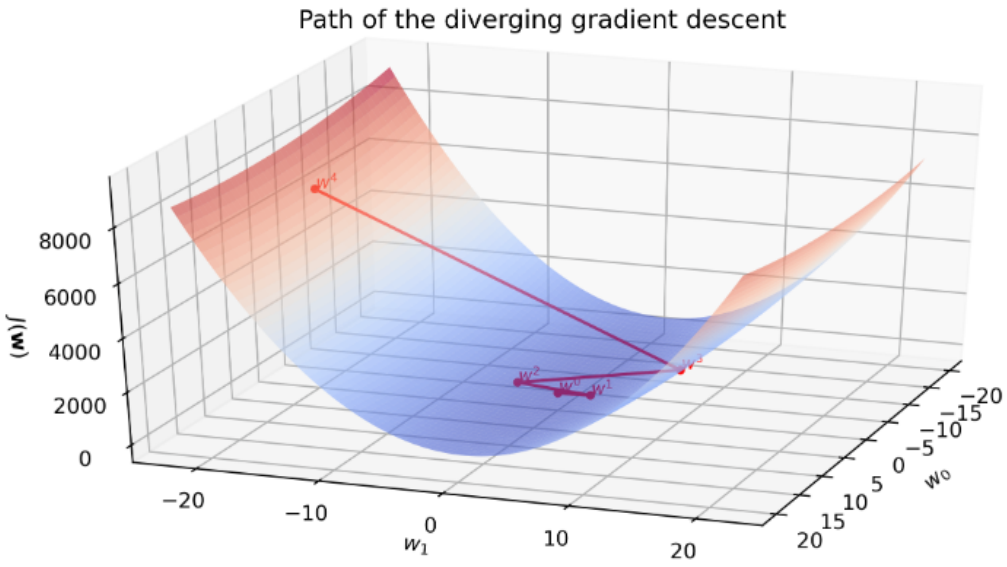


Figure 4.4: *Gradient Descent Algorithm.*

4.3 Metrics

Performance metrics play a crucial role in evaluating the effectiveness and accuracy of machine learning models. They provide insights into a model's predictive capabilities and help measure its performance across various tasks [14]. Importance:

- Performance metrics quantify the quality of predictions made by machine learning models.
- They provide actionable insights for model evaluation, comparison, and improvement.

4.3.1 Recall

The true positive rate (TPR), or the proportion of all actual positives that were classified correctly as positives, is also known as recall or sensitivity.

Recall is mathematically defined as:

$$\text{Recall (or TPR)} = \frac{\text{correctly classified actual positives}}{\text{all actual positives}} = \frac{TP}{TP + FN} \tag{6}$$

False negatives are actual positives that were misclassified as negatives, which is why they appear in the denominator. Lets take as an example the problem of the spam classification. Recall takes in account the percentage of the spam mails that were correctly classified as spam. Importance in Recall is given is some cases where False Negative are crucial. As an example in the case of detection of life threatening issues a False Positive case could have implications. That is the reason of seeking in minimizing this metric. But designing a system with the goal of minimizing the False Negatives cases could have a result in increasing the False Positives of the model in a try to minimize False Negatives. So **Recall** is used along the other metrics in order to increase the insight of the model structure and performance.

4.3.2 Accuracy

Classification accuracy is the most used and easily understood metric for evaluating the performance of a classification model, providing a quick insight of how well the model is performing in terms of valid predictions. This is calculated as the ratio of correct predictions to the total number of input Samples.

$$\text{Accuracy} = \frac{\text{No. of correct predictions}}{\text{Total number of input samples}} \tag{7}$$

It works great if there are an equal number of samples for each class. For example, we have a 90% sample of class A and a 10% sample of class B in our training set. Then, our model will predict with an accuracy of 90% by predicting all the training samples belonging to class A. If we test the same model with a test set of 60%

from class A and 40% from class B. Then the accuracy will fall, and we will get an accuracy of 60%.

Classification accuracy is good but it gives a False Positive sense of achieving high accuracy. The problem arises due to the possibility of mis-classification of minor class samples being very high. Also a model trying to achieve high accuracy in an a dataset that is imbalanced could classify all the samples to the dominant class.

4.3.3 Precision

Precision is the proportion of all the model's positive classifications that are actually positive.

$$\text{Precision} = \frac{\text{correctly classified actual positives}}{\text{everything classified as positive}} = \frac{TP}{TP + FP} \tag{8}$$

Precision is frequently used alongside recall (also call by sensitivity). Recall quantifies the model's ability to correctly identify actual positive cases. In many practical applications, finding the right balance between precision and recall is critical. To achieve this balance, metrics such as the F1 score are employed. The F1 score provides a harmonic mean of precision and recall, offering a single measure that takes both metrics into account. This approach is particularly useful in scenarios where a trade-off between precision and recall needs to be optimized, allowing for a more comprehensive evaluation of a model's performance.

4.3.4 F1-Score

F1 score is another machine learning evaluation metric that takes in account the predictive skill of a model by elaborating on its class-wise performance rather than an overall performance as done by accuracy. F1 score embodies two competing metrics precision and recall scores of a model, leading to its extensive use in recent research.

The accuracy metric measures how many times a model made a correct prediction across the entire dataset. This can be a robust metric only if the dataset is class-balanced; that is, each class of the dataset has the same number of samples.

Precision and recall offer a trade-off, one metric comes at the cost of another. More precision uses a harsher critic (classifier) that doubts even the actual positive samples from the dataset, thus lowering the recall score. On the contrary, more recall requires a careless critic that allows any sample that seems the same as a positive class to pass, which makes border-case negative samples categorized as "positive," and in this way lowering the precision. Preferably, the goal is the maximization of both precision and recall metrics in order to get the perfect classifier.

The F1 score combines precision and recall employing their harmonic mean, and maximizing the F1 score implies on the same time maximizing both precision and

recall. Thus, the F1 score is the preferable choice of researchers for evaluating their models in conjunction with accuracy.

The F1 takes values between (0,1). This range shows that the 1 is the best score achieved, meaning that the 1 is the perfect combination pf the precision and recall.

$$F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{9}$$

4.3.5 Confusion Matrix

Confusion Matrix is another great metric especially needed in multiclass classification. This metric sheds light to the combination of correct predictions and the actual prediction happened. This metric shows the actual accuracy in every class. There are four categories of this metric:

- **True Positive:** The amount of times our actual positive values are the same to the predicted positive. The model predicted a positive value, and it is correct in its prediction.
- **False Positive:** The amount of times our model falsely forecast negative values as positives. The model predicted a negative value, and it is positive.
- **True Negative:** The number of times our actual negative values are the same to predicted negative values. The model predicted a negative value, and it is actually negative.
- **False Negative:** The amount of times our model falsely forecast negative values as positives. The model predicted a negative value, and it is positive.

The tool of confusion matrix showcase the different results of predictions and actual results.

4.4 ResNet

Residual Network is a deep Learning model used for computer vision applications. The ResNet (Residual Neural Network) architecture was introduced by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian sun in their paper titled “Deep residual Learning for Image Recognition “ in 2015.

As the number of layers in the neural network grow the gradients of the loss function with respect to the weight may become very small during the process back-propagation. This makes it difficult for the latter layers to learn any pattern of importance as the updates of the weights are almost really small to have an impact.

Instead in some cases the gradients can become very large during back-propagation leading to instability in the training process by causing the weights to be updated by large amount, making the optimization process difficult to control. The degradation problem in neural networks refers to the phenomenon where, as the depth

of a neural network increases ,the performance of the network on the training data saturates and then starts to degrade [15].

4.4.1 Key Components of ResNet Architecture

This pre-trained model is consisted by the Residual blocks which are the main building blocks of Residual Neural network. In a traditional neural network, like the ones tried first in this thesis, the input is transformed by a set of convolutional layers then it is feeded to the activation function. On the contrary in a residual network the input to the block is aggregating to the output of the block thus creating a residual connection. The output of the residual block H(x) can be represented by H(x) = F(x) + x, F(x) represents the residual mapping learned by the network .The presence of identity term x allows the gradient to flow more easily Figure 4.5.

Stacking multiple residual blocks together finally creates these ResNet procedures. Using these multiple residual blocks together makes the ResNet architecture have the ability to be built **very deep**. This is something desired in this particular project because the EEG signals have complex patterns that are required to be learned by the deep architecture of this pre-trained model. The versions of ResNet with 50,101,152 layers were used.

A typical ResNet architectures uses **Global Average Pooling** as the final layer before the fully connected layer. Global Average Pooling reduces spatial dimensions to a single value per feature map providing a compact representation of the entire feature map which is also required when using data with large dimensions like the one used here.

In this diagram Figure 4.6 is shown the VGC-19 ,34 layer plain network and 34 layered residual network.

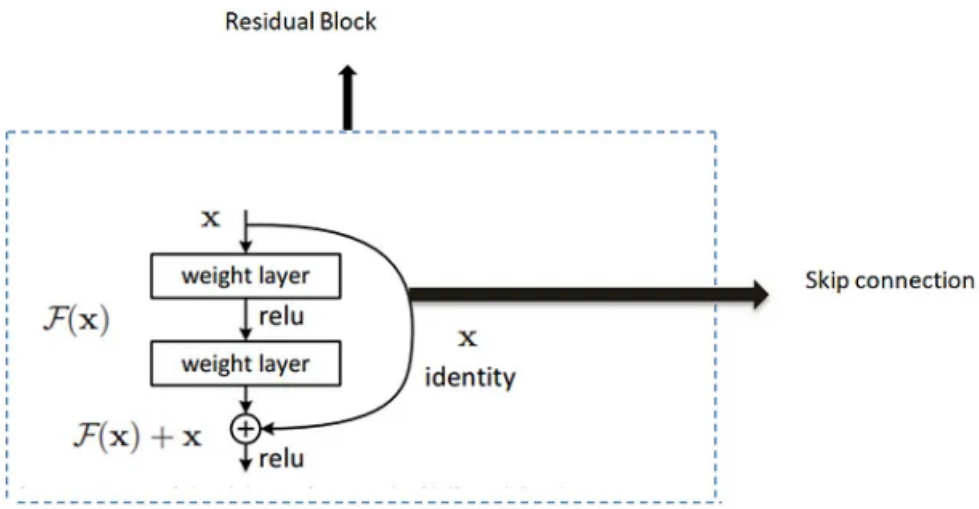


Figure 4.5: *Residual Block.*

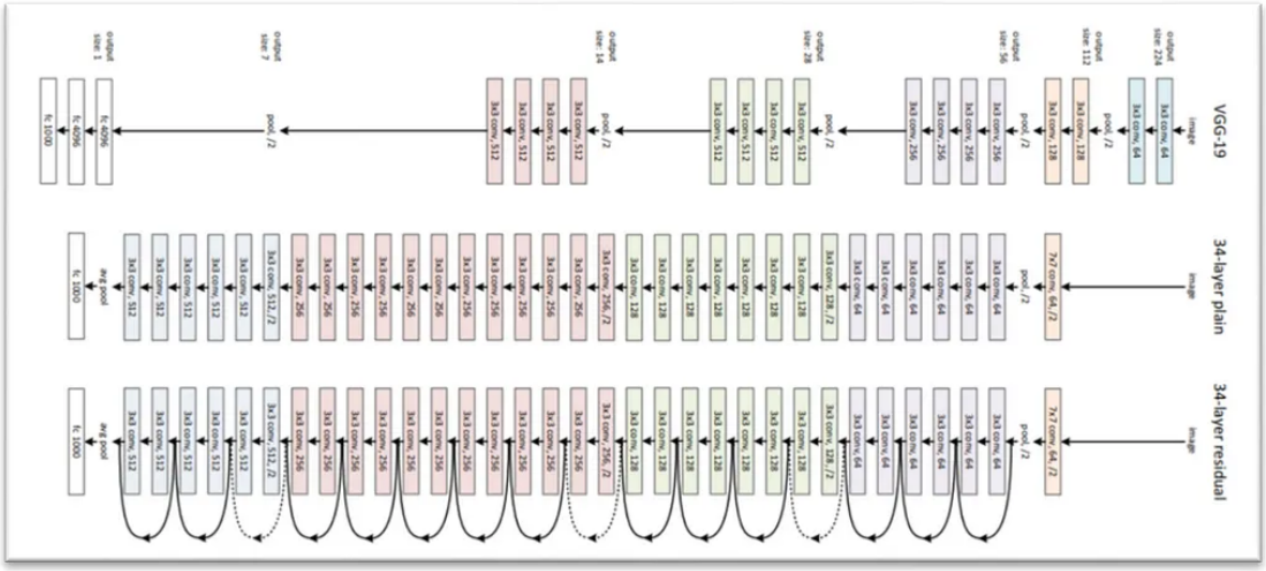


Figure 4.6: *34 layered Residual Neural Network.*

4.5 EfficientNet

EfficientNet uses a technique called compound coefficient to scale up models in a simple but effective manner. On the contrary of randomly scaling up width, depth or resolution, compound scaling uniformly scales each dimension with a certain fixed set of scaling coefficients. Using the scaling method and AutoML, the authors of efficient developed seven models of various dimensions, which exceeded the state-of-the-art accuracy of most convolutional neural networks, and with greater efficiency [16].

The compound scaling method is based on the idea of balancing dimensions of width, depth, and resolution by scaling with a constant ratio. The insight for the networks is, if the input image is bigger, then the network needs more layers to increase the receptive field and more channels to capture more better-quality patterns on the bigger image.

4.5.1 Key Components of EfficientNet Architecture

EfficientNet is based on the baseline network developed by the neural architecture search using the AutoML MNAS framework. The network is adjusted for obtaining the greatest accuracy but is also punished if the network is really heavy. It is also punished for slow inference time when the network spares too much time to make predictions. The architecture uses a mobile inverted bottleneck convolution.

This pre-trained model uses as initial layer a standard convolution followed by a batch normalization and a ReLU activation function which usually delivers the

best results. Inside its architecture there is a Convolution with 32 filters, kernel size of size 3×3 and stride of 2. Moreover, inside the deep layers there is a series of MBConv blocks employing different configurations that are used to capture the complex patterns of our specific dataset. Each of these blocks have depth wise separable convolutions and layers.

In the end of this model there is a final convolutional block which is then followed by the Global Average Pooling.

Chapter 5

Data Analysis and Results

5.1 Pre-processing

The data used in this project are electroencephalogram (EEG) signals visualized as spectograms. A spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time. These EEG signals were obtain via the BCI experiment as mentioned before and were transformed to spectograms which is essentially photos.

The information contained in these photos(spectograms) must be extracted first and be transformed to an appropriate form. This task was carried out by the first step of this thesis that was the pre processing part.

As it is easily understood the volume of the data(photos) is huge in order to be handled locally. The data, 33 pilots, were initially uploaded in a Google Drive space as zipped files. Then the pipeline of the Data Handling begins. The zipped files are unzipped in the cloud and then are saved into a Pandas Dataframe and regex patterns were used to extract metadata (pilot ID, motion type, event index, and EEG channel) from file paths. This function creates a pandas DataFrame Figure 5.1, providing a structured representation of the dataset. This class implements the following features:

- Image loading with the option of cropping (585x793 to 540x682) ensuring centering the original image and dropping any white spaces and resizing (default 128x128 or 224x224 for pre-trained models) Figure 5.2

- Label encoding for the four motion types (none: 0, left: 1, right: 2, both: 3). As seen also in the Figure 5.3 there are the four categories of the classes of the data used. This is also a good validation check that all the data is equally present and there in lack of data of any class.

After the data is saved with its full information into a Dataframe, data splitting is occurring dividing the original Dataset into 70% of the original dataset to train dataset, 15% validation and 15% testing respectively. This function allows for easy

configuration of batch sizes, shuffling, and number of worker processes for data loading. This is shown better in the Figure 5.4.

There are two **config.py** file that contains the necessary information for the pipeline to begin and the dataloading to occurs.

```
[6]  df = pd.DataFrame([extract_features_from_filepath(filepath) for filepath in file_list])
     df.head()
```

	pilot	motion	event	channel	filepath	extension
0	1	both	4	F1	data/Pilot1/both/event4/Pilot1_channelF1.jpg	.jpg
1	1	both	4	PO7	data/Pilot1/both/event4/Pilot1_channelPO7.jpg	.jpg
2	1	both	4	F3	data/Pilot1/both/event4/Pilot1_channelF3.jpg	.jpg
3	1	both	4	F2	data/Pilot1/both/event4/Pilot1_channelF2.jpg	.jpg
4	1	both	4	Fpz	data/Pilot1/both/event4/Pilot1_channelFpz.jpg	.jpg

Figure 5.1: *Quick inspection of the Dataframe.*

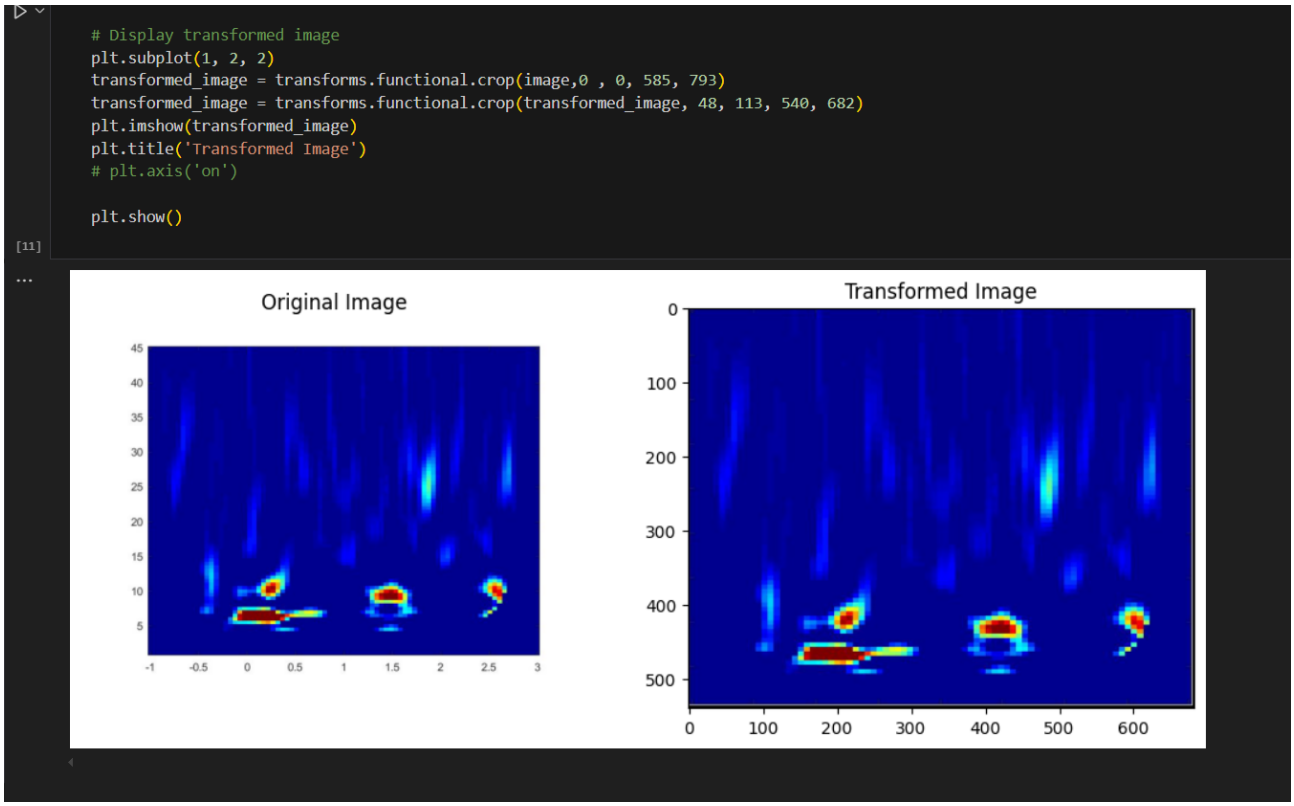


Figure 5.2: *Original vs Transformed image.*

5.2 Training Process

The training process is implemented in the train_and_eval_model function, which coordinates the entire training loop. Key features of the training process include:

- **Model initialization**. The pipeline supports both **custom models (cnn, cnn_large)** and **pre-trained models (ResNet50 or EfficientNet-B0)** with options for fine-tuning or feature extraction
- **Optimizer and loss function**: The code uses between **Adam**, **SGD**, and **RMSProp** optimizers with configurable **learning rate** and **weight Decay-L2 Regularization**. **Cross-Entropy Loss** is used as the criterion, suitable for multi-class classification.
- **Training Loop:** Each epoch involves a full pass through the training data, with gradients computed and applied for each batch. The model's performance is evaluated on the validation set after each epoch.
- **Early Stopping:** The training implements an early stopping mechanism to prevent overfitting, with a configurable patience period and minimum improvement threshold (delta). The parameter used to check for no improvement was set to 5 epochs Figure 4.2.

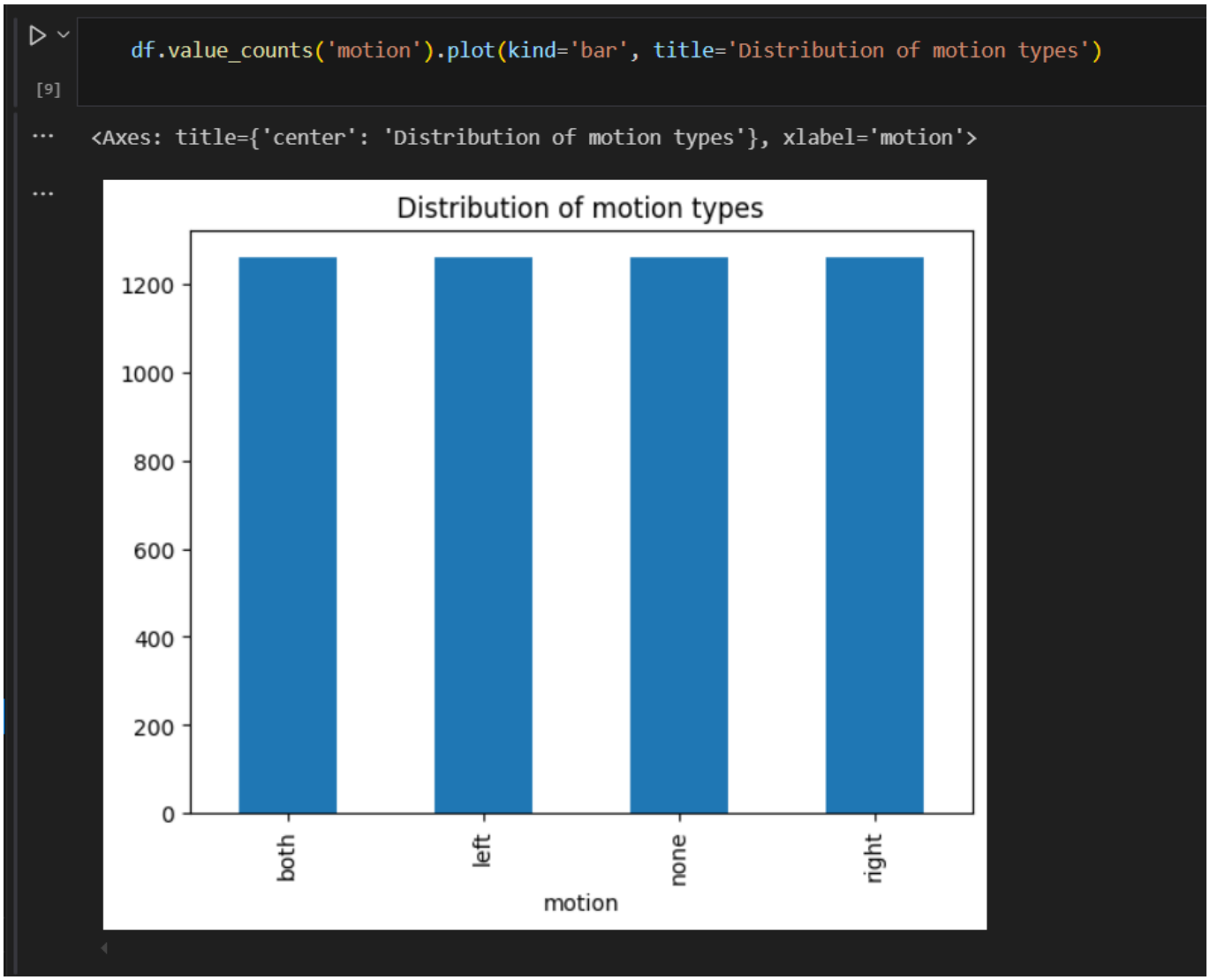


Figure 5.3: *Showcase of the 4 different classes.*

- **TensorBoard integration:** Training and validation metrics (loss and accuracy) are logged using TensorBoard, allowing for real-time monitoring and post-training analysis.
- **Model Checkpointing:** The best-performing model (based on validation loss) is saved during training.
- **Testing and Evaluation:** The testing phase is handled by the test_model function, which evaluates the trained model on the test set.
- **Hyperparameters:** This script uses itertools to generate combinations of hyperparameters (learning rates, weight decay values, and potentially different model architectures). It runs multiple training sessions with different configurations, allowing for a systematic exploration of the hyperparameter space.

```python
def get_dataloders(config, mocked=False):

    if mocked:
        datasetClass = SpectrogramDatasetMocked
    else:
        datasetClass = SpectrogramDataset

    df  = load_df()
    # Create a DataLoader for each split
    train_loader, val_loader, test_loader = get_train_val_test_loaders(df,
                                                                       dataloader_config_dict=config,
                                                                       datasetClass=datasetClass,
                                                                       verbose=True)
    dataloaders = {
        'train_loader':train_loader,
        'val_loader': val_loader,
        'test_loader': test_loader,
    }

    return dataloaders
```

Figure 5.4: *Script of the dataloader.*

- **Dropout:** as mentioned before is the rate defined by a percentage of neurons to be dropped in order to prevent over fitting. The rates were defined to 20% to 50% for the Custom Large CNN and EfficientNet.

In the Figure 5.5 is visible the main component of the pipeline that coordinates the training of this model. This script initially chooses the hardware between the use of **CPU** or **GPU**. The use **GPU** is necessary because the volume of data overloads the fewer and more sophisticated cores of CPUs. After that, the loading of the data is happening using the load_df() function. Once the Data is loaded then it is separated to three categories necessary for the procedure, the training, the validation and the test. As their name implies the **training set** is used for the training of the data, the **validation set** is used for the validation of the model under training and the **test set** is used for the final check of the model.

5.2.1 Model Architectures

The project uses multiple model architectures for the classification task, including custom CNN models and pre-trained models. This approach allows for a comparison between different architectures and utilizes transfer learning techniques.

- **Custom CNN(SpectrogramClassifier in cnn.py)**. It is a relatively simple convolutional neural network designed for spectrogram classification. The code of the CNN is visible in Figure 5.6. Key features include:
  - **Input**: 3-channel images (RGB representations of spectrograms)
  - Two convolutional layers (Conv2d) with ReLU activation and max pooling.
  - Two fully connected layers for classification
  - **Output**: 4 classes corresponding to none, left, right, and both limb movements

- **Larger Custom CNN (SpectrogramClassifier in cnn_large.py)**. The code of the larger CNN is visible in Figure 5.7
  - **Input**: 3-channel images (RGB representations of spectrograms)
  - Four convolutional layers with increasing filter sizes (10, 20, 40, 80)
  - Batch normalization after each convolutional layer for improved training stability
  - Leaky ReLU activations instead of standard ReLU, potentially helping with the vanishing gradient problem
  - Dropout layers for regularization (0.25 and 0.5 dropout rates)
  - Four fully connected layers with decreasing sizes (4096, 1024, 256, output_dim)

- **Pre-trained Models:**The project also uses pre-trained models, specifically ResNet50 and EfficientNetB0. These models are implemented in the get_pretrained_model function in utils.py. Key points:
  - Models are loaded with pre-trained weights from ImageNet
  - The final classification layer is modified to output 4 classes
  - Option to freeze the backbone (basically not optimizing the pre-trained weights of the feature extraction module included in the pretrained model) or fine-tune the entire network

The structure of this projects has the following advantages:

- **Flexibility:**The system allows easy switching between custom and pre-trained models through configuration settings.
- **Transfer Learning:**By using pre-trained models, the project leverages knowledge from large-scale image datasets, which can be particularly useful based on the EEG dataset size.
- **Hyperparameter Tuning:**The train_multiple.py script facilitates systematic exploration of hyper-parameters, including learning rates and weight decay values. This is important for optimizing model performance.
- **Adaptability to Input Size:**The custom models are designed to handle different input sizes, with the larger CNN potentially better suited for higher resolution spectrograms.
- **Regularization Techniques:**The use of dropout and early stopping(5 epochs) in the larger CNN model helps in preventing over-fitting, which is crucial in medical signal classification tasks where datasets might be limited in size(20% to 50% for the Custom Large CNN and EfficientNet).

5.3 Training and Evaluation Process

These features are employed in order for the successful implementation of this task:

- **Data Handling:** The use of PyTorch's DataLoader with custom Dataset classes ensures efficient data feeding during training.
- **Optimization:** The Adam optimizer is used with configurable learning rate and weight decay, allowing for fine-tuning of the optimization process.
- **Loss function:** Cross-Entropy Loss is employed, which is appropriate for multi-class classification tasks.
- **Performance Monitoring:**The integration with TensorBoard allows for real-time tracking of training and validation metrics, facilitating early detection of issues like overfitting or underfitting.
- **Model Selection:** The best model is saved based on validation performance, ensuring that the final model is the one that generalizes best to unseen data.
- **Early Stopping:** This mechanism prevents overfitting by halting training when the validation performance stops improving. The parameter for this check was set to 5 epochs.
- **Unbiased Evaluation:** The separate test set evaluation provides an unbiased estimate of the model's performance on completely unseen data.

In order to evaluate the model, the following metrics were employed, accuracy, F1-Score, precision and recall. The most important of course is the accuracy, which is translated to how many correct predictions it did.

5.4 Results

Initially, the custom-made CNN-model was employed and trained. The results were not promising, and even after Hyper-Parameter tuning the **Test Accuracy** was around 25% . This indicated that as the results were near the probability of random assigning an image to a class, our models where not complex enough to capture important information. The next step was to try a larger CNN which ought be more capable of capturing all the information of these spectograms. However even the larger CNN with more hidden layers did not succeeded of surpassing the previous percentage.

The next step was to employ a state of the art pre trained model which are embedded in the libraries used. These models are trained in huge datasets of millions of photos and make use of multiple hidden layers. The accuracy percentage of these models nearly doubled the scores of the custom made models which is more than

reasonable if someone consider the complexity and ingenuity of theses models. After some Hyper-Parameter tuning the optimum parameters and variables were found.

In the following sections the results will be quoted for better understanding. Per model used will be shown two tables, a table with all the characteristics of the model and another table with the metrics achieved and the respective **Hyperparameters**.

5.4.1 Custom Convolutional Neural Network(CNN)

In this Table 5.1, is shown the general technical specifications of the **Custom Small CNN**. As it is evident it contains only two layers, employs ReLu activation function and Max Pooling. The training of this model include an iteration of the hyperparameters **Optimizer**, **Learning Rate** and **Weight Decay**. The criterion used for the training was the **Cross Entropy Loss**, that meaning that the model's target was to minimize this value. In the Table 5.2 there are presented the best results for this model Table 5.1. The best performance(Accuracy) is observed for **Learning Rate = 0.005**, **Weight Decay = 0** and **ADAM optimizer**. Also the **Early Stopping** method worked and stopped the training process in the 10th epoch preventing unnecessary training. Along with the **Learning Rate** there are experiments iterating between values of **Weight Decay**. These values are (0, 0.1, 0.01, 0.001, 0.0001). In all of the experiments the better results were delivered when **Weight Decay =0**. Also there are presented in the following tables the results using the **Optimizers SGD, RMsprop** Table 5.3 and Table 5.4. The fact that the best **Weight Decay** is 0 means that the risk of overfitting is not present in our dataset. Also in Figure 5.8 is shown the **Confusion Matrix** of this model. This matrix depicts the correlation between the **Predicted Classes** and **Actual Cases** meaning how many cases were classified correctly. We could see that this model performed overall better in the Class 4(Both).

Component	**Description**
Model Type	Convolutional Neural Network (CNN)
Input	Spectrogram (1 channel)
Convolutional Layers	2
Fully Connected Layers	2
Activation Function	ReLU
Pooling	Max Pooling (kernel size 2)
Output Classes	4 (none, left, right, both)
Layer Details	
conv1	Input: 1, Output: 6, Kernel: 3x3
conv2	Input: 6, Output: 12, Kernel: 3x3
fc1	Input: 10800, Output: 128
fc2	Input: 128, Output: 4
Stride	Default (equal to kernel size(2))

Table 5.1: *Summary of SpectrogramClassifier custom CNN Architecture*

Accuracy Train	Accuracy Test	Epoch	Learning Rate	Weight Decay
31.21	22.45	15	0.05	0
27.12	22.86	10	0.01	0
34.94	**24.45**	**10**	**0.005**	**0**
32.06	21.29	16	0.001	0
26.56	20.98	16	0.0001	0
16.81	19.56	18	0.05	0.1
25.27	20.69	12	0.01	0.1
27.34	21.31	16	0.005	0.1
27.60	20.59	18	0.001	0.1
23.64	19.34	11	0.0001	0.1

Table 5.2: *Experiment Results using ADAM optimizer with the following columns: Accuracy Train(%) Accuracy Test(%), Learning Rate, Weight Decay, Optimizer for the Small CNN.*

Accuracy Train	Accuracy Test	Epoch	Learning Rate	Weight Decay
23.25	18.12	25	0.05	0
21.32	20.23	23	0.01	0
14.34	12.51	21	0.005	0
22.36	18.92	18	0.001	0
16.64	15.82	22	0.0001	0
22.14	17.61	28	0.05	0.1
17.21	14.61	26	0.01	0.1
17.44	15.12	24	0.005	0.1
22.06	19.90	19	0.001	0.1
22.31	21.50	18	0.0001	0.1

Table 5.3: *Experiment Results using SGD optimizer with the following columns: Accuracy Train(%) Accuracy Test(%), Learning Rate, Weight Decay, Optimizer for the Small CNN.*

5.4.2 Custom Large Convolutional Neural Network(CNN)

In this Table 5.5, is shown the general technical specifications of the **Custom Large CNN**. As it is evident it contains now four layers, employs Leaky ReLu activation function and Max Pooling. The training of this model include an iteration of the hyperparameters **Optimizer**, **Learning Rate** and **Weight Decay**. The criterion used for the training was also the **Cross Entropy Loss**, that meaning that the model's target was to minimize this value. In the Table 5.6 there are presented the best results for this model. The best performance(Accuracy) is observed for **Learning Rate = 0.01**, **Weight Decay = 0** and **ADAM optimizer**. As in the previous case **Weight Decay** delivered best results when it was equal to 0. The explanation that the best **Weight Decay** is 0 it also means that the risk of overfitting is not present in our dataset. Also there are presented in the following

Accuracy Train	Accuracy Test	Epoch	Learning Rate	Weight Decay
21.52	20.12	17	0.05	0
22.12	20.30	17	0.01	0
18.14	16.19	19	0.005	0
21.61	19.12	24	0.001	0
19.41	14.21	23	0.0001	0
19.40	16.16	28	0.05	0.1
21.98	18.13	21	0.01	0.1
17.41	16.24	19	0.005	0.1
19.65	17.10	25	0.001	0.1
20.10	19.40	17	0.0001	0.1

Table 5.4: *Experiment Results using RMsprop optimizer with the following columns: Accuracy Train(%) Accuracy Test(%), Learning Rate, Weight Decay, Optimizer for the Small CNN.*

tables the results using the **Optimizers SGD, RMsprop** Table 5.7 and Table 5.8. In addition in Figure 5.9 is shown the **Confusion Matrix** of this model. This matrix depicts the correlation between the **Predicted Classes** and **Actual Cases** meaning how many cases were classified correctly. We could see that this model performed slightly better Class 4(Both). But also we could observe that the Class 1(None) is also close. We draw the conclusion that the classes Right or Left have some special features that could be distinguished by this model.

Component	Description
Model Type	Convolutional Neural Network (CNN)
Input Channels	3
Convolutional Layers	4
Fully Connected Layers	4
Activation Function	Leaky ReLU
Pooling	Max Pooling (kernel size 2)
Normalization	Batch Normalization after each conv layer
Regularization	Dropout (0.25 and 0.5)
Output	Variable (output_dim parameter)
Layer Details	
conv1	Input: 3, Output: 10, Kernel: 3x3
conv2	Input: 10, Output: 20, Kernel: 3x3
conv3	Input: 20, Output: 40, Kernel: 3x3
conv4	Input: 40, Output: 80, Kernel: 3x3
fc1	Input: 2880, Output: 4096
fc2	Input: 4096, Output: 1024
fc3	Input: 1024, Output: 256
fc4	Input: 256, Output: output_dim
Stride	Default (equal to kernel size(2))

Table 5.5: *Summary of SpectrogramClassifier custom Large CNN Architecture*

5.4.3 Custom Pre-Trained Model(EfficientNet-B0)

In this Table 5.9, is shown the general technical specifications of the **EfficientNet**. This pre-trained model employs Sigmoid Linear Unit-Swish(SiLU) activation function and its depth varies. The training of this model include an iteration of the hyperparameters **Optimizer**, **Learning Rate** and **Weight Decay**. The criterion used for the training was also the **Cross Entropy Loss**, that meaning that the model's target was to minimize this value. In the Table 5.10 there are presented the best results for this model. The best performance(Accuracy) is observed for **Learning Rate = 0.005**, **Weight Decay = 0** and **ADAM optimizer**. In these pre-trained models there is another customizable parameter **No-freeze-backbone**. No-freeze-backbone is a feature of the pre trained models which allows to freeze or not some weights of the deep learning model. This allows for shorter time of training. In this task there were both cases tested no layer freezed and layers freezed. This essentially allowed the final classification layers to be fine-tuned on our task, bringing knowledge from the prior training into both feature extraction layers, hidden layers as well as the classification layers. This parameter was found to deliver the best results when it had the value "NO", meaning that every layer was trained. This makes sense as this data were specific and needed every information from the pre-trained model. The fact that the best **Weight Decay** is 0 it also means that the risk of overfitting is not present in our dataset. Also there are presented in the

Accuracy Train	Accuracy Test	Epoch	Learning Rate	Weight Decay
50.74	**25.49**	**10**	**0.01**	**0**
29.41	25.02	10	0.05	0
35.45	24.43	10	0.005	0
50.23	24.56	12	0.001	0
29.56	22.98	13	0.0001	0
34.25	21.93	13	0.05	0.1
27.23	23.78	13	0.01	0.1
32.51	22.31	14	0.005	0.1
48.63	23.73	18	0.001	0.1
28.34	22.26	17	0.0001	0.1

Table 5.6: *Experiment Results using ADAM optimizer with the following columns: Accuracy Train(%), Accuracy Test(%), Epoch, Learning Rate, Weight Decay, Optimizer for the Large CNN.*

Accuracy Train	Accuracy Test	Epoch	Learning Rate	Weight Decay
48.10	21.40	18	0.01	0
28.48	24.21	28	0.05	0
37.52	25.30	22	0.005	0
51.30	24.60	24	0.001	0
26.80	22.90	24	0.0001	0
33.20	22.30	25	0.05	0.1
29.30	24.80	23	0.01	0.1
33.50	23.34	19	0.005	0.1
51.60	24.10	29	0.001	0.1
38.31	24.45	17	0.0001	0.1

Table 5.7: *Experiment Results using SGD optimizer with the following columns: Accuracy Train(%), Accuracy Test(%), Epoch, Learning Rate, Weight Decay, Optimizer for the Large CNN.*

following tables the results using the **Optimizers SGD, RMsprop** Table 5.11 and Table 5.12.

In addition in Figure 5.10 is shown the **Confusion Matrix** of this model. This matrix depicts the correlation between the **Predicted Classes** and **Actual Cases** meaning how many cases were classified correctly. We could see that this model performed equally good in all classes. In this pre trained model the best predictions are made in the **Right, Left Classes**. This could be a product of better understanding of the spectograms, as this model had been trained in millions of photos and could distinguishes better some artifacts. In the Figure 5.11 is shown the training process for the best model(EfficientNet-B0) which is stopped in the 13 epoch. This could be understood better in Figure 5.12 where is shown how the **Loss** is gradually decreasing meaning that the model training finds the best hyperparameters.

Accuracy Train	Accuracy Test	Epoch	Learning Rate	Weight Decay
44.20	22.20	12	0.01	0
38.40	23.50	21	0.05	0
37.90	24.31	17	0.005	0
52.30	24.56	12	0.001	0
38.29	21.80	13	0.0001	0
34.50	23.30	14	0.05	0.1
31.30	22.50	13	0.01	0.1
37.10	20.30	14	0.005	0.1
49.80	23.71	23	0.001	0.1
29.40	21.60	19	0.0001	0.1

Table 5.8: *Experiment Results using RMsprop optimizer with the following columns: Accuracy Train(%), Accuracy Test(%), Epoch, Learning Rate, Weight Decay, Optimizer for the Large CNN.*

Characteristic	**Description**
Model	EfficientNet-B0
Type	Compound Scaled CNN
Depth	Variable (scaled)
Input Size	224 x 224 x 3
Main Building Block	Mobile Inverted Bottleneck ConvBlock
Squeeze-and-Excitation	Yes
Activation Function	Swish (SiLU)
Batch Normalization	Yes
Output	1000 classes (ImageNet)
Parameters	5.3 million

Table 5.9: *Summary of EfficientNet-B0 Architecture*

5.4.4 Custom Pre-Trained Model(ResNet)

In this Table 5.13, is shown the general technical specifications of the **Res-Net**. This pre-trained model employs ReLu activation function and its depth varies. The training of this model include an iteration of the hyperparameters **Optimizer**, **Learning Rate** and **Weight Decay**. The criterion used for the training was also the **Cross Entropy Loss**, that meaning that the model's target was to minimize this value. In the Table 5.14 there are presented the best results for this model. The best performance(Accuracy) is observed for **Learning Rate = 0.005**, **Weight Decay = 0** and **ADAM optimizer**. In these pre-trained models there is another customizable parameter **No-freeze-backbone**. As happened in the previous presented pre-trained model this parameter was found to deliver the best results when it had the value "NO", meaning that every layer was trained. This makes also sense. The fact that the best **Weight Decay** is 0 it also means that the risk of overfitting is not present in our dataset. In addition in Figure 5.10 is shown the

AccTrain	AccTest	Epoch	Learning Rate	Weight Decay
80.12	55.98	15	0.05	0
82.81	56.61	13	0.01	0
84.72	**56.78**	**13**	**0.005**	**0**
83.65	56.34	14	0.001	0
79.87	52.28	17	0.0001	0
78.32	52.81	13	0.05	0.1
80.21	52.11	19	0.01	0.1
82.22	54.83	16	0.005	0.1
82.38	53.64	16	0.001	0.1
77.71	51.83	19	0.0001	0.1

Table 5.10: *Experiment Results using ADAM optimizer with the following columns: Accuracy Train(%), Accuracy Test(%), Epoch, Learning Rate, Weight Decay, Optimizer for the EfficientNet-B0 Architecture and No-Freeze-Backbone.*

AccTrain	AccTest	Epoch	Learning Rate	Weight Decay
79.84	55.12	24	0.05	0
83.95	51.89	18	0.01	0
82.93	53.42	18	0.005	0
81.28	52.71	18	0.001	0
80.53	50.46	16	0.0001	0
78.65	53.93	17	0.05	0.1
84.47	50.34	28	0.01	0.1
83.45	51.67	25	0.005	0.1
81.14	53.82	17	0.001	0.1
75.93	50.65	19	0.0001	0.1

Table 5.11: *Experiment Results using SGD optimizer with the following columns: Accuracy Train(%), Accuracy Test(%), Epoch, Learning Rate, Weight Decay, Optimizer for the EfficientNet-B0 Architecture and No-Freeze-Backbone.*

Confusion Matrix of this model. This matrix depicts the correlation between the **Predicted Classes** and **Actual Cases** meaning how many cases were classified correctly. We could see that this model performed also equally good in all classes. Also there are presented in the following tables the results using the **Optimizers SGD, RMsprop** Table 5.11 and Table 5.16. In this pre trained model the best predictions are made in the **Both, Right Classes**. But the dataset was small so there is no conclusion drawn.

AccTrain	AccTest	Epoch	Learning Rate	Weight Decay
82.45	54.92	14	0.05	0
84.65	55.84	15	0.01	0
86.34	55.91	12	0.005	0
85.28	55.67	13	0.001	0
81.56	51.98	16	0.0001	0
80.15	52.23	14	0.05	0.1
82.34	51.87	18	0.01	0.1
84.18	54.12	15	0.005	0.1
84.52	53.21	17	0.001	0.1
79.83	51.42	18	0.0001	0.1

Table 5.12: *Experiment Results using RMsprop optimizer with the following columns: Accuracy Train(%), Accuracy Test(%), Epoch, Learning Rate, Weight Decay, Optimizer for the EfficientNet-B0 Architecture and No-Freeze-Backbone.*

Characteristic	**Description**
Model	ResNet50
Type	Residual Network
Depth	50 layers
Input Size	224 x 224 x 3
Convolutional Layers	49
Fully Connected Layers	1
Skip Connections	Yes
Batch Normalization	Yes
Activation Function	ReLU
Output	1000 classes (ImageNet)
Parameters	25.6 million

Table 5.13: *Summary of Res-Net Architecture*

5.4.5 Overall

In the previous tables were presented the best models(Accuracy) with its respective hyperparameters. The best Accuracy was achieved from the EfficientNet pretrained model with **Learning Rate = 0.005**, **Weight Decay = 0**, for **No-freeze-backbone**. Also really close was the ResNet model achieving similar results. The least successful models, were the custom CNN models.

More specifically the top 3 models of the Custom Large CNN, EfficientNet and ResNet where chosen in the Table 5.17 to be shown along with other metrics that were clalculated in this step.

AccTrain	AccTest	Epoch	Learning Rate	Weight Decay	Freeze-Backbone
80.12	55.98	15	0.05	0	No-freeze-backbone
90.13	54.42	23	0.01	0	No-freeze-backbone
91.53	55.77	22	0.005	0	No-freeze-backbone
92.77	55.09	9	0.001	0	No-freeze-backbone
96.57	43.9	7	0.0001	0	No-freeze-backbone
94.53	**56.5**	**8**	**0.0005**	**0**	**No-freeze-backbone**
78.22	53.18	19	0.05	0	freeze-backbone
88.87	52.24	21	0.01	0	freeze-backbone
89.31	54.11	27	0.005	0	freeze-backbone
91.72	52.23	10	0.001	0	freeze-backbone
94.51	45.92	13	0.0001	0	freeze-backbone
93.34	54.13	12	0.0005	0	freeze-backbone

Table 5.14: *Experiment Results using ADAM optimizer with the following columns: Accuracy Train(%), Accuracy Test(%), Epoch, Learning Rate, Weight Decay, Optimizer for the ResNet Architecture and employing Freeze-Backbone.*

AccTrain	AccTest	Epoch	Learning Rate	Weight Decay	Freeze-Backbone
82.34	54.87	24	0.05	0	No-freeze-backbone
92.45	53.98	22	0.01	0	No-freeze-backbone
93.68	54.92	23	0.005	0	No-freeze-backbone
94.82	54.67	20	0.001	0	No-freeze-backbone
98.23	43.45	18	0.0001	0	No-freeze-backbone
96.78	55.84	12	0.0005	0	No-freeze-backbone
80.45	52.76	18	0.05	0	freeze-backbone
90.92	51.89	22	0.01	0	freeze-backbone
91.45	53.87	26	0.005	0	freeze-backbone
93.85	51.98	15	0.001	0	freeze-backbone
96.73	45.23	17	0.0001	0	freeze-backbone
95.52	53.76	18	0.0005	0	freeze-backbone

Table 5.15: *Experiment Results using SGD optimizer with the following columns: Accuracy Train(%), Accuracy Test(%), Epoch, Learning Rate, Weight Decay, Optimizer for the ResNet Architecture and employing Freeze-Backbone.*

```python
def run_complete_training_pipeline(config=None, dataloaders={}):

    if not config:
        config = utils.class_from_module(config_global_module)

    if config.device == "auto":
        device = "cuda:0" if torch.cuda.is_available() else "cpu"
    else:
        device = config.device
    print(f"Running on:", device)

    train_loader = dataloaders["train_loader"]
    val_loader = dataloaders["val_loader"]
    test_loader = dataloaders["test_loader"]

    if config.pretrained_model:
        model = utils.get_pretrained_model(model_name=config.pretrained_model_name, num_classes=4,
                                           freeze_backbone=config.freeze_backbone)
    else:
        MyModelClass = utils.dynamically_load_model_from_settings(config=config)
    # criterionClass = utils.dynamic_import(config.criterion)

        model = MyModelClass()
    criterion = nn.CrossEntropyLoss()#criterionClass()
    optimizer = torch.optim.Adam(lr=config.learning_rate,
                                 weight_decay=config.weight_decay,
                                 params=model.parameters())

    print(model)
    print(criterion)
    print(optimizer)

    model.to(device)
    criterion.to(device)

    log_dir = utils.create_and_return_name_of_current_log_dir(config=config)
    writer = SummaryWriter(log_dir=log_dir)

    utils.save_experiment_to_tensorboard(config=config, writer=writer)

    train_and_eval_model(model, epochs=config.epochs, train_loader=train_loader, val_loader=val_loader,
                         optimizer=optimizer, criterion=criterion, device=device, writer=writer)

    test_model(model=model, test_loader=test_loader, criterion=criterion, device=device,
               writer=writer)

    writer.close()
```

Figure 5.5: *Script of the main training of the pipeline.*

training > models > cnn.py > ...

```python
import torch
import torch.nn as nn

class SpectrogramClassifier(nn.Module):
    def __init__(self):
        super(SpectrogramClassifier, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, kernel_size=3)  # input channels: 1 (mono), output channels: 6, kernel size: 3
        self.conv2 = nn.Conv2d(6, 12, kernel_size=3)
        self.fc1 = nn.Linear(10800, 128)  # input features: 2880, output features: 128
        self.fc2 = nn.Linear(128, 4)  # output classes: 4 (none, left, right, both)

    def forward(self, x):
        x = torch.relu(nn.functional.max_pool2d(self.conv1(x), 2))  # max pool with kernel size 2
        # print(x.shape)
        x = torch.relu(nn.functional.max_pool2d(self.conv2(x), 2))
        # print(x.shape)
        x = torch.flatten(x, 1)
        # x = x.view(-1, 269724)  # flatten the output of conv2
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# next_layer_size = ((input_size - kernel_size + 2 * padding ) / stride ) + 1
```

Figure 5.6: *Script of CNN code.*

AccTrain	AccTest	Epoch	Learning Rate	Weight Decay	Freeze-Backbone
80.23	53.89	13	0.05	0	No-freeze-backbone
89.34	52.87	20	0.01	0	No-freeze-backbone
91.45	53.92	19	0.005	0	No-freeze-backbone
93.27	53.45	9	0.001	0	No-freeze-backbone
96.89	42.98	7	0.0001	0	No-freeze-backbone
95.45	54.67	8	0.0005	0	No-freeze-backbone
78.92	51.98	16	0.05	0	freeze-backbone
88.45	50.87	21	0.01	0	freeze-backbone
90.23	52.65	24	0.005	0	freeze-backbone
92.54	51.23	10	0.001	0	freeze-backbone
95.67	44.12	11	0.0001	0	freeze-backbone
94.28	52.98	12	0.0005	0	freeze-backbone

Table 5.16: *Experiment Results using RMsprop optimizer with the following columns: Accuracy Train(%), Accuracy Test(%), Epoch, Learning Rate, Weight Decay, Optimizer for the ResNet Architecture and employing Freeze-Backbone.*

Metric	EfficientNet	ResNet	CNN-Large
F1-Score	0.55	0.49	0.40
Precision	0.58	0.36	0.29
Recall	0.55	0.48	0.34
Accuracy	56.78%	56.5%	25.49%

Table 5.17: *Best Models of each category*

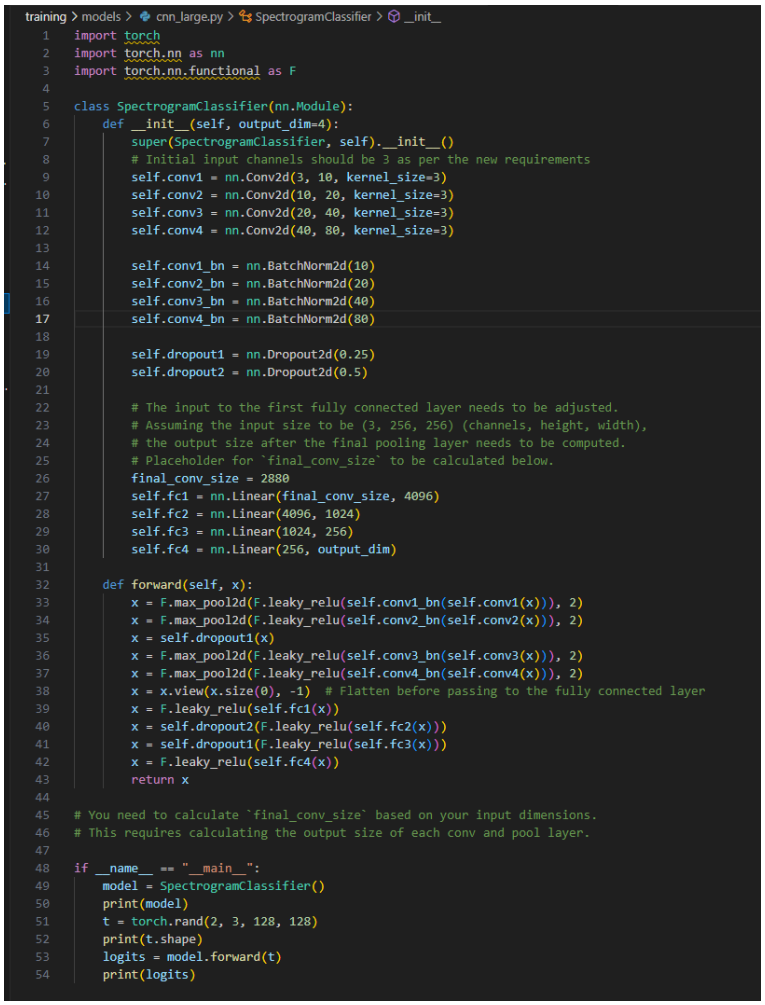


Figure 5.7: *Script of the Large CNN code.*

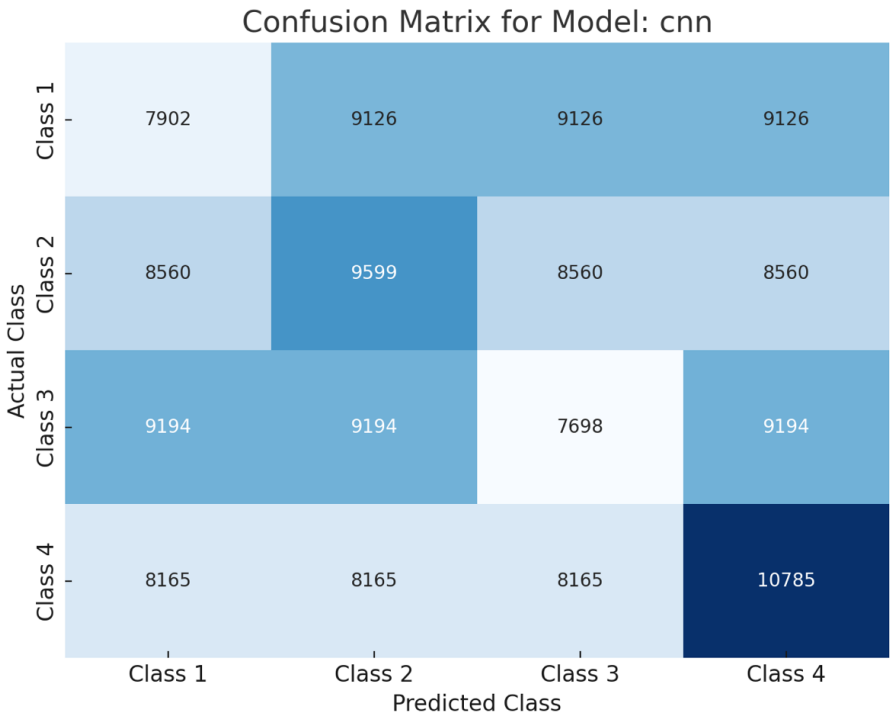


Figure 5.8: *Confusion Matrix for Custom CNN model.*

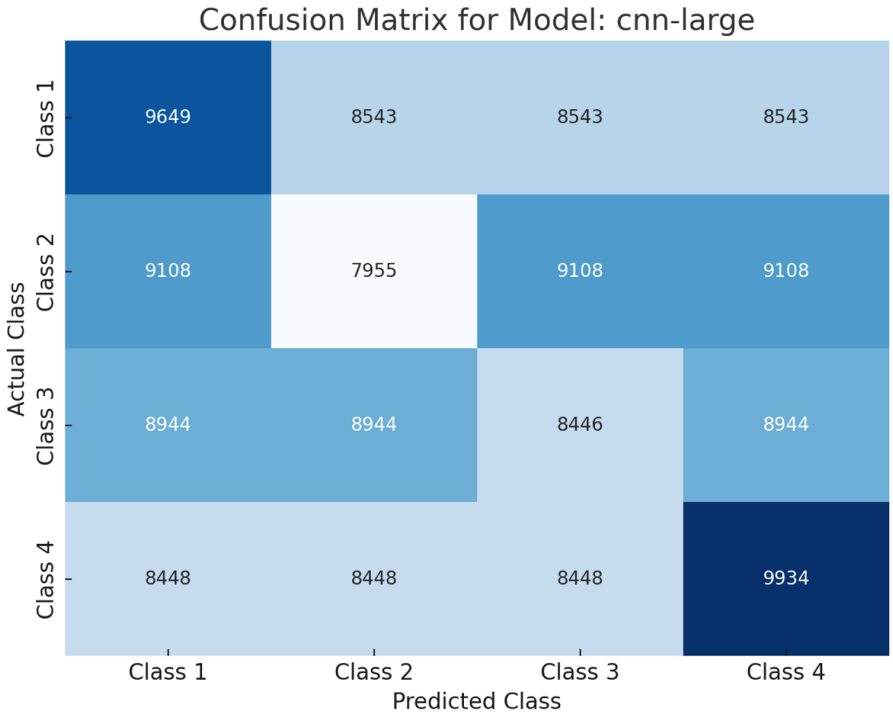


Figure 5.9: *Confusion Matrix for Custom Large CNN model.*

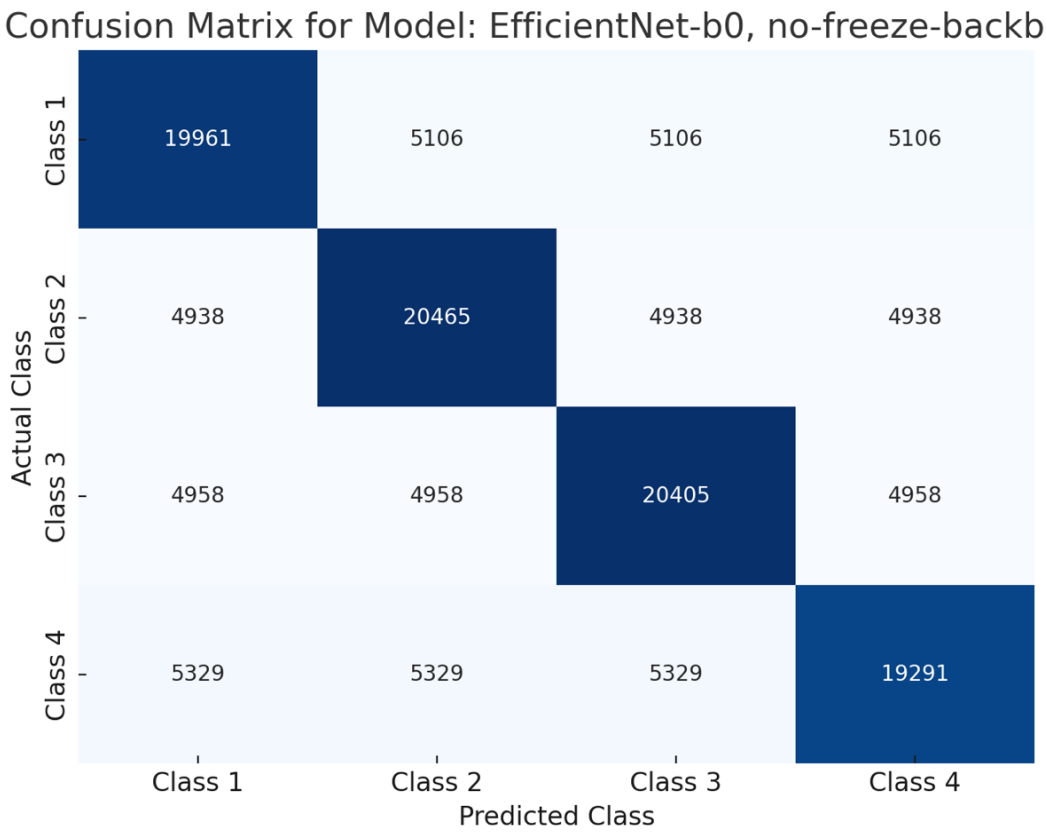


Figure 5.10: *Confusion Matrix for EfficientNet-B0 model.*

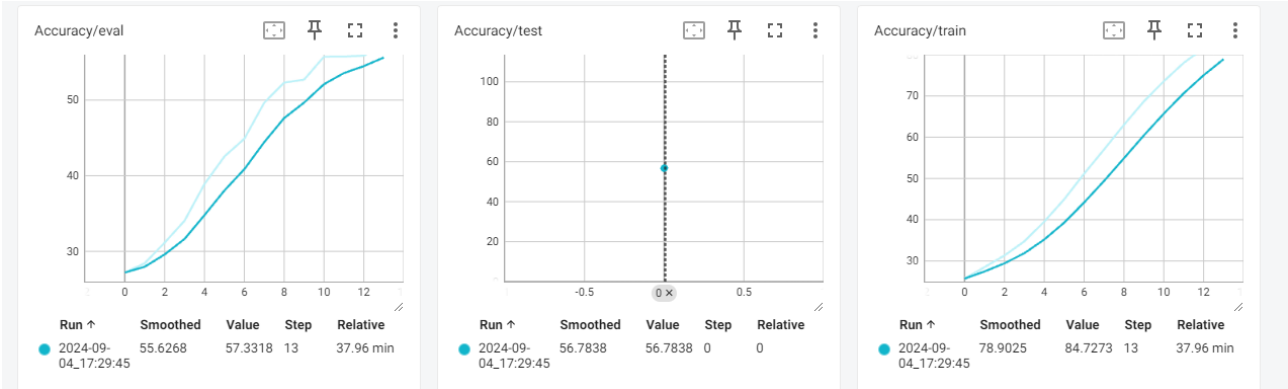


Figure 5.11: *Accuracy for EfficientNet-B0 model during the training phase.*

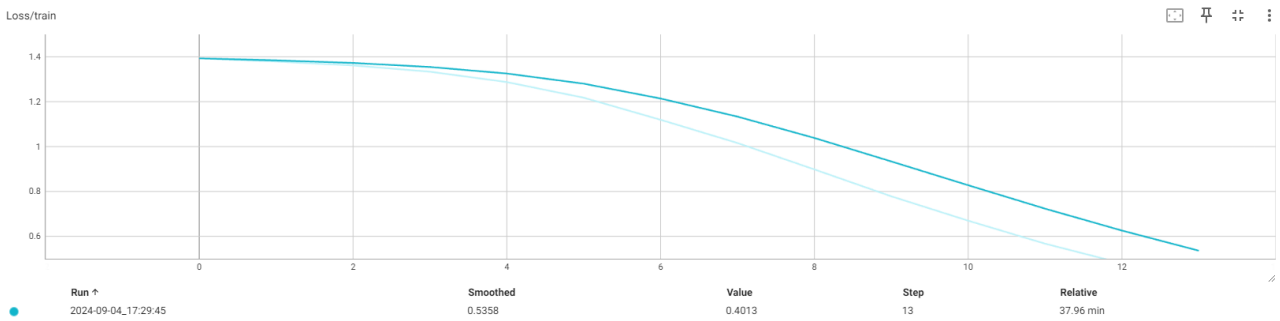


Figure 5.12: *Graphic representation of Loss for EfficientNet-B0 model during the training phase.*

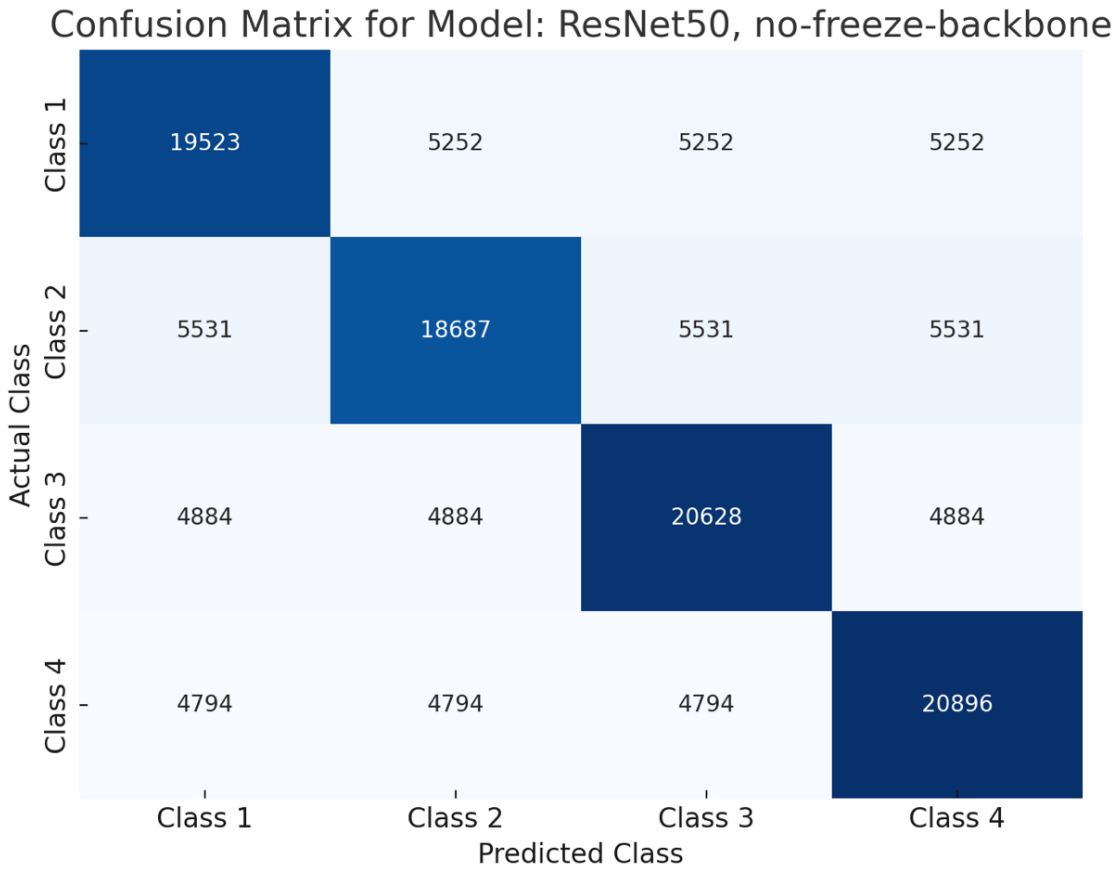


Figure 5.13: *Confusion Matrix for ResNet model.*

Chapter 6

Conclusion & Future Work

6.1 Conclusion

As this is thesis is concluding we will summarize some key aspects and conclusions drawn by this task. This experiment utilizes theoretical background of many scientific fields that have been around since the last half of the previous century. More specific this experiment employed EEG signals which have been a product of the Neuroscience field and methods and theory of some field initially seemed irrelevant to this task, the field of Deep Learning and Machine Learning. As evident by the results several important conclusions can be drawn regarding both the effectiveness of different models and the inherent challenges of the task. One of them was of course the combination of two different scientific fields. In this experiment have been employed various and different between them Deep Learning architectures, ranging from self designed Custom Designed CNNs and up to state-of-the-art pre-trained models trained and destined for other uses. All these different architectures which have been used are showing different characteristics in their approach to classifying limb movements from EEG spectrograms.

This thesis was structured by firstly making an introduction about the type of task and challenge that would be tackled, then by presenting all the theoretical background someone would need to effectively follow and understand the theory that lies between the lines of this thesis, after that the related work that helped and inspired this thesis and of course in the end all results are shown, presented and interpreted in order to extract some conclusions regarding this task, the data used and the methods of theory employed. In the next paragraphs will be showcased the comparative analysis of the experiment.

The comparison of the results extracted by the various methods, showed that the pre-trained architectures had a really clear advantage. The **EfficientNetM-model** was the best in class, achieving the astonishing accuracy of 56.78%, but also succeeding in getting the highest F1-score of 0.55, Precision of 0.58, and Recall of 0.55. The also pre-trained ResNet architecture/model gave also metrics like the accuracy close to that of the EfficientNet model, accomplishing 56.5% accuracy close to it. The results of these pre trained models outperformed, by far, the two custom

CNNs. These two custom CNNs could hardly exceed the accuracy rate of 25%. This somewhat led to the belief that the advanced feature extraction capabilities of the pre-trained models had been pivotal in decoding the generic patterns which are existing in various EEG spectrograms. Optimization methods and techniques revealed also some interesting patterns across various the architectures.

In this experiment one of the key aspects that were tried and tested were the use of **optimizers**. The correct choosing of an optimizer can have great impact in terms of performance and speed of training. The ADAM optimizer always outperformed both the other two optimizers. The other two optimizers were the SGD and RMSprop tried for all the models employed. This led to the result, that its superior capability in finding patterns in the landscape of loss of this particular issue greatly assisted in the successful implementation of this problem. Generally from the bibliography too the **ADAM Optimizer** is well known for getting the best results out of the others. Out the three optimizers the SGD Optimizer is an effective tool but simple in terms of complexity. With this optimizer every task is beginning in order to establish a base line for results. We thought that this optimizer would be of great assistance with our problem because we had noisy data but we failed to get the best results with this. After that we tested the RMSprop optimizer which was achieving similar results. Eventually the use of the ADAM gave the best results of all. ADAM combines the advantages of the other optimizers. The case in which it differs is when it employs the momentum which can help to not get stuck in some local maxima and minima. As also evident by our data with the SGD optimizer the training genrally lasted longer.

The issue of sensitivity to the key word of **Learning Rate** was significantly different between self designed-customized and pre-trained models. This hyperparameter is really important in terms of training and optimization procedure. Essentially this hyperparameter decree the size of the steps that the model is changing during the training which is the try to find the optimal loss. If the learning rate takes greater sizes it helps land quicker in a result but may result in high overshoot and risks to miss it. On the contrary a smaller rate could impose slower training. The best performance achieved could be ensured for the self designed custom CNNs when the learning rate was higher for example for values of 0.01. On the other hand for the pre-trained models, higher performances were obtained for the lower rates like 0.005 and 0.0005. This inconsistency in performance shows essential differences in the optimization dynamics of these 2 inherently different architectural methods.

Next in line is the comparison of the results,in terms of **Regularization** hyperparameters is the Weight Decay. All models perform equally optimal when the values of the Weight Decay were equal to zero. This could lead us to believe that overfitting may have not been an issue in this particular task. This makes sense, given the dataset size(small) and complexity in the current experiment. Also that means that this very regularization method was not needed as it did not improve the metrics. This might change when the datasets varies in terms of volume. The fact that the best performance of pre-trained models happened without the method network freezing also led us believe the assumption that the model needs to adapt

its whole feature architecture to the specific characteristics of EEG signals and data.

The most critical part in Machine Learning model is the data used as input. Their quality and variance could be the difference between a successful model that can generalize from the given data and a failed model that performs relatively well in the trained data but fails to replicate the results in unknown data(overfitting). The experiments and this very **pipeline**, demonstrates the ability of the model to handle the **EEG signals** and succeed in the Limb movement classification. The high density and complexity of the EEG signal could be dealt with the use of Convolutions and Deep Learning. Another thing to consider is that we did not have spectograms but rather photos of them. This, practically meant that time was available in the photos but not in a way that could help, like a vector. That was the reason that the architectures used did not take in account the time. The use of architectures like RNN, LSTM were disregarded because they use aggregation in time domain.

In order for the data to be extracted from the EEG signals and used in a CNN architecture there was transformation of the EEG data to spectrograms which may have led to the elimination of important information. This restriction might be imposed upon all the models tried and tested in this very task leading to a limit of improvements of the specific dataset and data representation. The maximum accuracy exhibited is rather low and for the pre-trained model is at best, mediocre. This is caused possibly by small dataset of the EEG signals, information loss during the transformation to spectograms and of course we should take in account the inherent difficulty to decode the brain mysterious structure and function.

Another thing evident from the tables presented with the hyperparameter tuning is that the two **custom CNN models** exhibits low **Accuracy Training** and it makes sense that the **Accuracy Test** would be also low. On the other hand the **Pre-Trained** models scores high Accuracy Train but fails to generalize on the Test Data. This means that the optimal model is the pre-trained model but needs improvements in terms of dataset. The difference in performance between training and testing in pre-trained models points towards the ability of these architectures to learn from the training data effectively, but generalization remains confined to the quality and quantity of the available data. Nevertheless, this pipeline exhibits great signs of robustness and potential for further improvements with the most appropriate data.

This experiment and the following study of the results has shown both the possibilities and current restrictions of deep learning methods for EEG-based movement classification. Whereas pre-trained models have achieved very good results based on the given dataset, the limit of performance metrics marked, makes the possibility of future better results really dependent in these two aspects. Firstly and mainly in data representation(EEG spectograms) and secondly in the model architecture. The significantly better and higher performance exhibited in the pre-trained models emphasize the value of transfer learning in this field but also points out the need of improvement of the feature extraction required for an effective EEG signal interpretation.

6.2 Future Work

As shown in this thesis there is much potential in furthering the collaboration of the field of the Neuroscience and Machine Learning but also some limitations and obstacles that were met. In order to get the most of these two fields and create some tools that would greatly help the society and people there is need of more research. This research could follow plenty of directions towards improving this specific pipeline and of course broaden it.

The first and most important part of the ML is the data itself. Increasing the volume of the data could help the layers of a Deep Learning to capture more effectively the complex patterns of an EEG signal. Also the quality of the data could increase in terms of, either better resolution of the spectograms or use of the signal in its raw form avoiding deterioration of its quality and complexity. Better resolution was not doable due to not enough memory available. There is the need for parallel training (multiple models training in parallel or more than 50Gb of RAM). Also, using spectrograms directly as np array rather than images, as images already compress (thus degrading information) the original signal.

After ensuring the existence of proper and perfect quality of data the next thing is the methodology and tools that could take advantage of these data. The pipeline of the classification could change completely logic. Instead of using 4 classes(left, right, both, none), there could two stages Binary-classification greatly improving the total accuracy. Firstly, the model could distinguish between move or not-move and then if there is movement could move to the final result. This approach is less complex and could return better results.

After establishing the need of appropriate data and methodology another key part is the utilization of already pre trained models trained specifically in **medical data**. This could further help ensuring that the medical data used are been processed by models specifically created for this type of data.

After using every available mean to ensure that the data has great quality and is handled properly the next thing is the training itself. To achieve this there could be a more **systematic hyperparameter tuning** resulting in further refinement of the model such as random, grid search etc. The thorough investigation of the best parameters for each model would deliver the best results and gets the best out of each practice and dataset.

Further pursuit devoted to unfold the mysteries of neural activities will not only improve the quality of Human Life but also greatly expand the fields of neuroscience, Biomedicine and of course Machine Learning, Maths and every aspect of it.

References

[1] T. B. N. Association, *The british neuroscience association/*, 2003. [Online]. Available: https://www.bna.org.uk/static/uploads/resources/BNA_English.pdf/.

[2] A. Author, *Title of web page*, Accessed: 2023-10-01, 2023. [Online]. Available: https://www.ibm.com.

[3] deep-learning-deep-neural-network, *The british neuroscience association/*, 2003. [Online]. Available: https://www.techtarget.com/searchenterpriseai/definition/deep-learning-deep-neural-network/.

[4] Appyhigh Technology Blog. «Convolutional neural networks: A brief history of their evolution». Accessed: [insert date here]. (2020), [Online]. Available: https://medium.com/appyhigh-technology-blog/convolutional-neural-networks-a-brief-history-of-their-evolution-ee3405568597.

[5] L. Alzubaidi, J. Zhang, A. J. Humaidi, *et al.*, «Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions», *Journal of Big Data*, vol. 8, no. 53, 2021.

[6] S. U. Khan, M. Majid, M. G. Linguraru, and S. M. Anwar, «Upper limb movement execution classification using electroencephalography for brain computer interface», in *2023 45th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, IEEE, 2023.

[7] R. Zhang, Y. Chen, Z. Xu, L. Zhang, Y. Hu, and M. Chen, «Recognition of single upper limb motor imagery tasks from eeg using multi-branch fusion convolutional neural network», *Frontiers in Neuroscience*, vol. 17, p. 1 129 049, 2023. DOI: 10.3389/fnins.2023.1129049.

[8] D. Zhang, L. Yao, K. Chen, S. Wang, X. Chang, and Y. Liu, «Deep learning for eeg motor imagery classification based on multi-layer cnns feature fusion», *Future Generation Computer Systems*, vol. 86, pp. 138–149, 2018.

[9] M. Xu, J. Han, Y. Wang, T.-P. Jung, and D. Ming, «Eeg-based lower-limb movement onset decoding: Continuous classification and asynchronous detection», *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 28, no. 9, pp. 1908–1916, 2020.

[10] O. P. Idowu, A. E. Ilesanmi, X. Li, O. W. Samuel, P. Fang, and G. Li, «An integrated deep learning model for motor intention recognition of multi-class eeg signals in upper limb amputees», *Computer Methods and Programs in Biomedicine*, vol. 206, p. 106 121, Jul. 2021. DOI: 10.1016/j.cmpb.2021.106121.

[11] IBM. «Regularization». Accessed: insert access date here. (2024), [Online]. Available: https://www.ibm.com/topics/regularization.

[12] Analytics Vidhya. «How does the gradient descent algorithm work in machine learning?» Accessed: insert access date here. (2020), [Online]. Available: https://www.analyticsvidhya.com/blog/2020/10/how-does-the-gradient-descent-algorithm-work-in-machine-learning/.

[13] D. P. Kingma and J. Ba, «Adam: A method for stochastic optimization», *arXiv preprint arXiv:1412.6980*, vol. 8, no. 53, 2014.

[14] GeeksforGeeks. «Metrics for machine learning model». Accessed: insert access date here. (2024), [Online]. Available: https://www.geeksforgeeks.org/metrics-for-machine-learning-model/.

[15] geeksforgeeks, *Geeksforgeeks*, 2023. [Online]. Available: https://www.geeksforgeeks.org/residual-networks-resnet-deep-learning/.

[16] M. Tan and Q. V. Le, «Efficientnet: Rethinking model scaling for convolutional neural networks», in *International Conference on Machine Learning*, 2019. [Online]. Available: https://arxiv.org/abs/1905.11946.