FNIRS DATA CLASSIFICATION FOR BRAIN COMPUTER INTERFACE USING DEEP LEARNING

AUTHOR

AHMAD SUBHANI

REGN NUMBER

115577

SUPERVISOR

DR. SYED OMER GILANI

DEPARTMENT OF BIOMEDICAL ENGINEERING AND SCIENCES

SCHOOL OF MECHANICAL & MANUFACTURING ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

SEPTEMBER 2019

fNIRS Data Classification for Brain Computer Interface Using Deep Learning

Author

Ahmad Subhani

Regn Number

00000115577

A thesis submitted in partial fulfillment of the requirements for the degree of

MS Biomedical Engineering

Thesis Supervisor:

Dr. Syed Omer Gilani

Thesis Supervisor's Signature: _____

DEPARTMENT OF BIOMEDICAL ENGINEERING AND SCIENCES

SCHOOL OF MECHANICAL & MANUFACTURING ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,

ISLAMABAD

SEPTEMBER 2019

Thesis Acceptance Certificate

It is certified that the final copy of MS Thesis written by Ahmad Subhani (Registration No. 00000115577), of SMME (School of Mechanical & Manufacturing Engineering) has been vetted by undersigned, found complete in all respects as per NUST statutes / regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS/MPhil Degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in this dissertation.

Signature: _____

Name of Supervisor: Dr. Syed Omer Gilani

Date: _____

Signature (HOD): _____

Date: _____

Signature (Principal): _____

Date: _____

FORM TH-4

National University of Sciences & Technology

MASTER THESIS WORK

We hereby recommend that the dissertation prepared under our supervision by: Ahmad Subhani (REG# 115577) Titled: "fNIRS data classification for Brain Computer Interface using Deep Learning" be accepted in partial fulfillment of the requirements for the award of MS degree. GRADE : __

Examination Committee Members

1. Name: Dr. Muhammad Jawad Khan Signature:_____

2. Name: Dr. Saima Zafar Signature:_____

3. Name: Dr. Umer Ansari Signature:_____

Supervisor's name: Dr. Syed Omer Gilani Signature:_____

Date:_____

_____ _____

Head of Department Date

COUNTERSINGED

Date: _____ _____

Dean/Principal

Declaration

I certify that this research work titled "*fNIRS data classification for Brain Computer Interface using Deep Learning"* is my own work. The work has not been presented elsewhere for assessment. The material that has been used from other sources it has been properly acknowledged / referred.

Signature of Student

Ahmad Subhani

00000115577

Plagiarism Certificate (Turnitin Report)

This thesis has been checked for Plagiarism. Turnitin report endorsed by Supervisor is attached.

Signature of Student
Ahmad Subhani
Registration Number
000001155777

Signature of Supervisor
Dr. Syed Omer Gilani

Copyright Statement

Acknowledgements

Glory be to Allah, the Exalted, with all His Praise; True Glory belongs to Allah Who is free of all faults, free of all needs!

I am immensely grateful to my supervisor Dr. Omer Gilani for his valuable time and guidance during this thesis, for correcting my direction and keeping me motivated in the times of need. I am especially thankful to my co-supervisor Dr. Jawad Khan and his team from Pusan National University for providing me with access to their excellent fNIRS data and for his valued guidance in the project as well as in content generation for exams and research paper.

I am thankful to my parents for their unwavering care and patience; for being the refuge I could always return to for: strength, inspiration, and motivation.

I am thankful to the department's faculty for their help during coursework and for their guidance in the projects.

Finally, I am very thankful to my dear friends Suhaib Sadiq, Ahmed Raza, Atif Sultan, Azeem Alvi, Zaeem Hadi, Zaid Ahsan Shah, Saad Habib and to Sarosh Bilal, Amna- and Sonia -Malik. Thank you for your valuable ideas, reviews and continuing emotional support.

ahmad subhani

Dedicated to the future of physically and mentally challenged, to their struggles and dreams.

Abstract

Brain Computer Interfaces (BCIs) translate recorded brain data directly to machine commands that can be used to control external devices. They are composed of three different functions i.e. recording of data from the brain, processing of data to recognize the intention of the subject and translation of the data into appropriate command for the machine being controlled. Functional near-infrared spectroscopy (fNIRS) is among one of the brain signal recording techniques which uses near-infrared spectroscopy (NIRS) for functional neuroimaging. It uses near-infrared light wavelengths (between 650 and 1000 nm) to measure the optical absorption changes of brain tissues. Use of fNIRS for BCIs limited because of slow hemodynamic response to stimulus, blood flow in scalp and undeveloped techniques for classifying signals.

In this thesis we train Convolutional Neural Networks to classify fNIRS signals for BCIs. These networks classify raw signals with more than 95% testing accuracy for cognitive and imagery tasks with upto five separate categories in less than two milliseconds, (dependent on the processing power available), thus showing promising improvement in current classification efficiency.

Table of Contents

List of Figures

List of Tables

CHAPTER 1: INTRODUCTION

1.1 Background, Scope and Motivation

Brain Computer interfaces refer to the interfaces that provide a communication channel between a brain and a computer. These interfaces translate recorded brain data to machine commands that can be used to control external devices[1], [2]. They usually perform three different functions i.e. recording of data from the brain, processing of data to recognize the intention of the subject and translation of the data into appropriate command for the machine the subject is controlling. Data for BCIs can be recorded by various techniques. FNIRS is a neuroimaging technique used to study brain activation and brain disorder treatments, it uses sources and detectors of near infrared wavelengths (650-1000nm) of light to detect changes in the brain tissue [3], [4]These Sources and Detectors are attached non-invasively on the surface of head where light passes through the skull and falls upon the brain tissue before being reflected to the detectors. The amount of light reflected to the detectors depends on the light absorption characteristics of hemoglobin which is indirectly dependent on brain activation. Thus, brain activation can be quantified using fNIRS measurement.

Brain Computer interfaces can be very useful to patients with neurological pathologies such as ALS, Locked-in syndrome in gaining control over their lives and being independent[1]. These interfaces allow the subjects to control devices bypassing their own internal neural pathways that may not be working e.g. in controlling a prosthetic limb using BCIs amputees can control the prosthetics and gain some function back of their limbs similarly other patients that have mobility issues can also control devices such as wheel chairs, Hospital Beds and other commonly used electrical devices. BCIs can also facilitate healthy subjects in improving their efficiency and allowing better control with added functions to commonly used devices.

An ideal BCI should be affordable and comfortable for the subject, it should be accurate and fast moreover it should be able to conduct complex tasks that involve a large number of commands. Required tasks vary in complexity from being very simple such as binary tasks with only two commands to very complex requiring dozens of actions to be conducted either in parallel or in sequence[5]. To cater these needs we need a reliable neuroimaging and robust classifiers.

According to current trends, BCIs mostly use EEG for fast detection but it is used for reactive tasks that require cues for the subject to start[6]. More research is required for active

command generating by the subject and for working with cognition-based tasks. Fnirs in comparison to EEG is a relatively new technique which has shown promising results for BCIs using both active commands and for cognition-based tasks since it uses variations in oxyhemoglobin and deoxyhemoglobin changes.[7]

1.2 Research Problem

Although its use has shown promising results for BCIs, fNIRS relies on the slow hemodynamic response which takes up to 17 seconds for the signal to reach maximum potential. This induces an inherent delay in decoding signals. In the past various feature extraction such as ARMAX modelling, Adaptive Independent component Analysis etc. and classification techniques such as Linear Discriminant Analysis, Support Vector Machines, Artificial Neural Networks have been used with Fnirs. These techniques rely on statistical features that need to be extracted before classification.[8]

Convolution Neural Networks (CNNs) have been shown to classify Neural signals with higher efficiency, some very recent papers have also shown promising results to classify fNIRS signals[9]. In this thesis we propose to improve the accuracy and timing of current classification techniques by using CNNs for Quinary classification including cognitive tasks since most locked in patients can’t generate motor imagery signals as efficiently as normal healthy patients.[10], [11]

CHAPTER 2: Literature Review

2.1 Understanding the Working of FNIRS and CNNs

In this thesis Convolution Neural Networks have been used to classify Fnirs data for Quinary (Five Categories) Classification. It is therefore necessary to understand the working of Fnirs and CNNs. This segment describes the fundamentals of fNIRS and CNNs

2.1.1 Functional Near-Infrared Spectroscopy (fNIRS)

2.1.1.1 Theory and Data Recording

fNIRS is the use of near-infrared spectroscopy (NIRS) for the purpose of functional neuroimaging to measure the optical absorption changes of tissues. It measures brain activation indirectly through the scalp relying on the change in reflected light from the brain tissue [7]. When a brain area is activated it causes increase in metabolic activity causing an increased cerebral blood flow in the respective area, since near infrared light's reflection is dependent on the amount of blood [12]it is incident upon therefore this increased blood flow changes the amount of reflected light towards the detector. fNIRS is cheaper than modalities like FMRI and PET scan but with lower spatial resolution[13], it is also less sensitive to motion than fMRI, so it reduces noise and is comfortable for children [3]. This also makes it more suitable for long measurement and tasks which require movements.

To measure brain activation subjects are made to wear caps on which probes of detector or emitter optodes are attached. Every emitter emits near-infrared wavelengths of light that pass through the human skull and are reflected to the detectors where blood oxygenation levels are quantified.

Figure 1.1 shows a typical configuration of a source and detector optodes and how they are attached as probes to a wearable cap. Each pair constitutes an fNIRS channel. Since the reflected light depends on the hemoglobin in the tissue, we can use modified Beer lambart Law to quantify changes in hemoglobin. [12], [14]

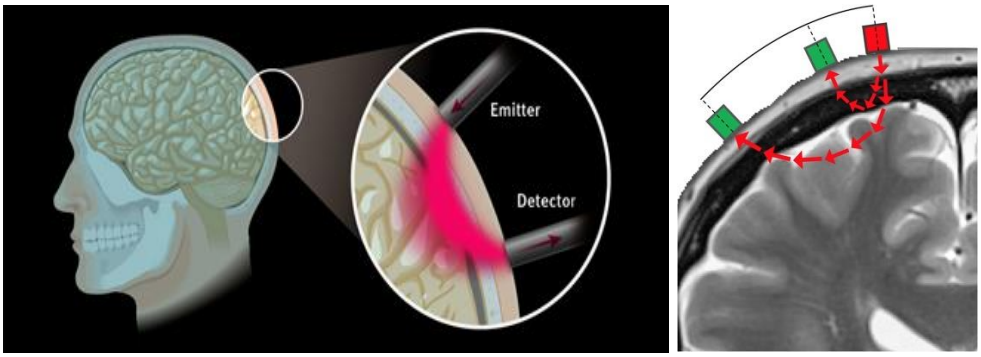


Figure 2.1: Placement of NIRS Emitters and Detectors on the scalp

Fnirs is a noninvasive technique for neuroimaging that can be used for Brain Computer Interfaces, other non-invasive modalities include Fmri and EEG. fNIRS in comparison to EEG and fMRI is easier and cheaper to record. It is also less prone to movement artifacts which makes it suitable for long term experiments. It also allows it to be commonly used with unstable subjects such as children.

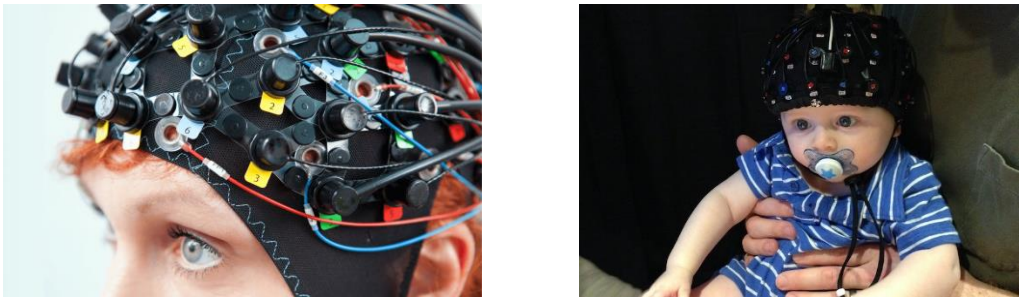


Figure 2.2: A typical cap worn on head over which these optodes are attached

The spatial resolution in detecting hemodynamic response of brain is better than detecting neuronal response to stimulus in EEG, but the temporal resolution is much lesser than EEG. These separate characteristics and the fact that they do not interfere with each other during simultaneous recording allows the use of both techniques together. Several studies under the name of Hybrid-BCIs record simultaneous data from EEG electrodes and Fnirs optodes. [13], [15]

2.1.1.2 Fnirs Signal

Although three types of NIRS devices can be used to record fNIRS signals. Continuous Wave - NIRS (CW-NIRS) devices are the most popular since they are simpler, cheaper and portable. They only measure the light attenuation so absolute value of hemoglobin concentration is not determinable[16] . Each channel configured using CW-NIRS uses values of molar coefficients and values of DPF to calculate concentration changes if change in hemoglobin concentration in tissue remains constant. This is not always true however precise quantification can be skipped as overall trends remain the same so a suitable combination of extinction coefficients and DPF can be chosen for concentration changes. An Fnirs signal can contain three types of noises that are often necessary to remove before further analysis.

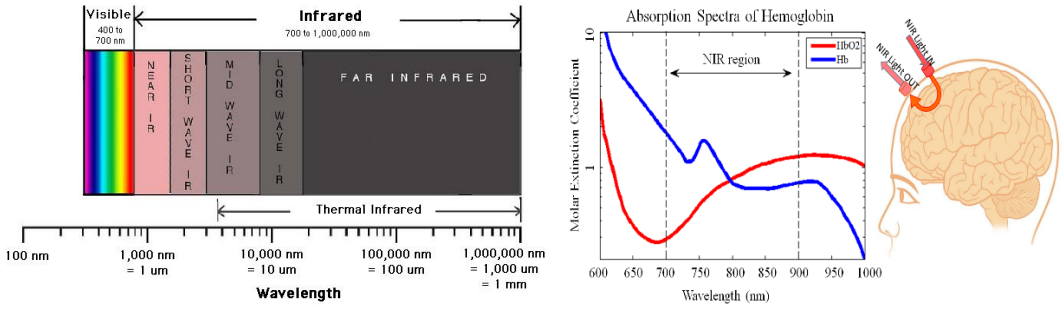


Figure 2.3: Light spectrum with Near IR just after the visible spectrum (Left) and Absorption Spectra of Hemoglobin and oxy-Hemoglobin (Right)

a. Motion Artifacts

Motion artifacts appear as sharp peaks and baseline shifts in the data due to movements of head during recording and unstable connection with the scalp, they are avoided at all costs and usually whole data is rejected if they are too many. In cases they can't be ignored e.g. when dataset is small or when they can't be avoided like recordings from children It is better to remove the artifacts and restore the signal. [8], [14], [17]

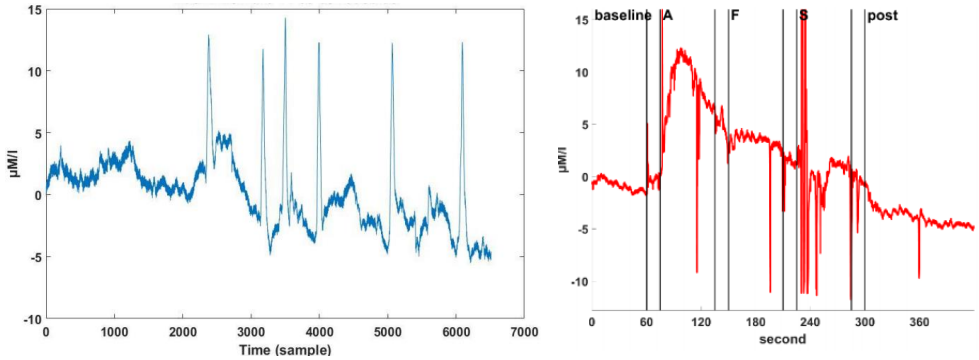


Figure 2.4: Spikes and Baseline shift artifacts shown left and right respectively

Several methods are used for removing spike motion artifacts such as recording additional data about movements of the subject using reference channels ; Principal Component Analysis (PCA) based on the fact that these artifacts appear in the form of peaks at the same time in several channels; techniques making use of the fact that motion artifacts are always positively correlated in Hb and HbO while normal data is always inversely correlated; and spline interpolation.

Detecting baseline shift due to motion is tricky since it very closely resembles hemodynamic response of the brain. Therefore, complex techniques are required to distinguish these as quickly as possible in BCIs.

b. Instrumental Noise

Instrumental noise is a random white noise which can be filtered using simple low-pass filtering techniques. Techniques such as block averaging, Moving Average and cutting off higher frequencies after data conversion to frequency domain are used. However, sensitivity of these techniques needs to be determined manually to avoid distorting data.[13], [18]

c. Physiological Noise

Physiological noise is added to fnirs signal due to several systemic physiological changes other than the hemodynamic response, activities such as changes in blood pressure, heart rate, respiration, Mayer wave, blood flow outside the brain beneath skull that are superficial layers.

These components of the signal known as physiological noise are defined by Schlokman as any non-functional brain activity. [13], [14], [18]

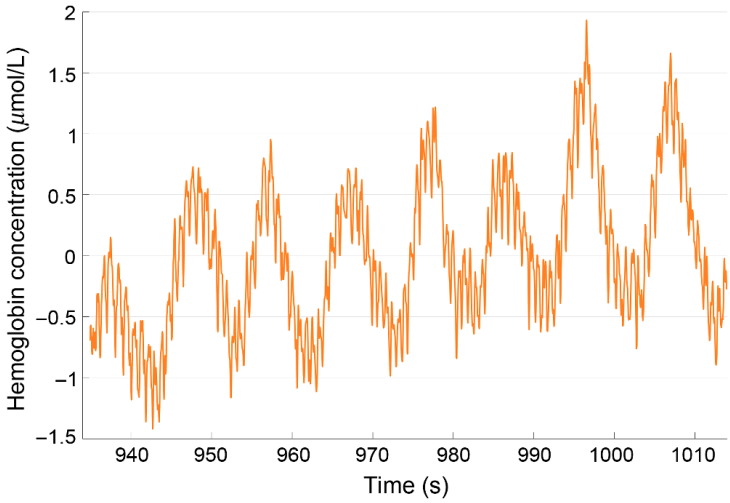


Figure 2.5: OxyHb oscillations at cardiac frequency (1 Hz) and at M-wave frequency (0.1 Hz) enhanced with deep breathing at 0.1 Hz.

Various ranges of lower thresholds of frequencies of a typical hemodynamic response have been reported in literature. For instance, lower cutoffs such as 0.5Hz (Keles, Barbour, & Omurtag, 2016; Yücel, Selb, Boas, Cash, & Cooper, 2014; Cui, Bray, Bryant, Glover, & Reiss, 2011) and 0.4Hz (Spichtig, Scholkmann, Chin, Lehmann, & Wolf, 2012).

i. For non-overlapping ranges

Higher frequency components can be removed using low pass filters, similarly very slow-moving components (0.01Hz or lesser) can be removed using high pass filters. Other common filtering techniques such as wavelet filtering, smoothing etc. can also be used.

ii. For Overlapping Ranges

Components with frequency range (0.01-0.5Hz) such as Respiration and Mayer wave overlap with the Fnirs signal therefore normal low-pass of high pass filters can't be used to remove these noises.

Several Methods are used e.g. method proposed by Zhang (2005) which uses information of physiological noise from in baseline data. Zhang proposes that physiological noise remains nearly same during baseline recording and during activity for the same subject so estimated noise from baseline can be removed from the active data recordings. Another method as discussed before in removing motion artifacts is the use of reference channels that are placed very close to each other (3cm), so the light does not reach brain are used. They record superficial components of data without being affected by brain activity thus providing an estimate of noise. In a similar fashion sometimes causes of physiological noise such as heart rate, respiration, and blood pressure are monitored with separate sensors along with fNIRS recording. Just like removing motion artifacts Principal Component analysis is used to remove estimated physiological noise from the data.

2.1.2 Convolution Neural Networks (CNNs)

A Convolutional Neural Network is a category of Artificial Neural Networks that does not require separately extracted features before classification. CNNs are designed in a grid like pattern inspired from an animal visual cortex to automatically learn important features present in the data at different spatial hierarchies through different layers such as convolution layers, pooling layers and fully connected layers. In very simple words a CNN finds what to find from comparing and optimizing results. [9], [19]

A typical CNN processes date in three types of layers most frequently connected in series namely Convolution Layer, Pooling Layer and a fully connected layer also known as Dense Layer. Before passing data to these layers the data is usually but not necessarily normalized and down sampled according to the complexity of required spatial features. The Layers are described in detail as follows

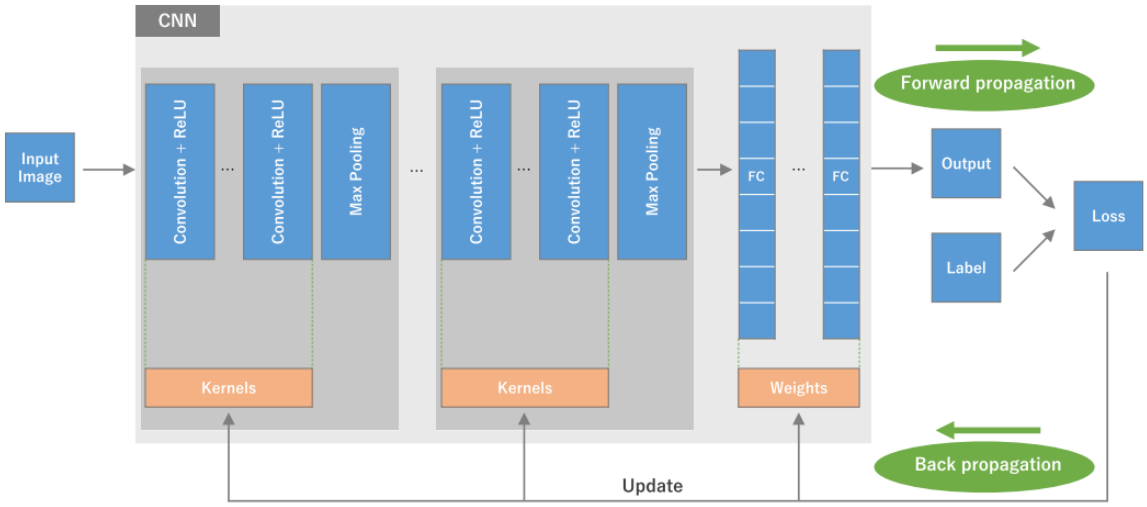
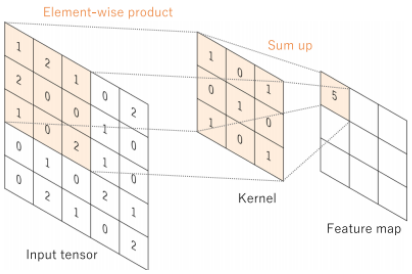


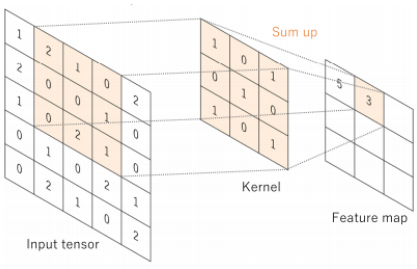
Figure 2.6: Flow chart showing a typical CNN [9], [20]

2.1.2.1 Convolution Layer and Activation

A convolution layer is made up of fixed number of convolution kernels that have a pre-defined size. These kernels are basically small matrices usually 2x2 or 3x3 with default starting values that are updated to extract features. These kernels are convolved with input i.e. an element wise product is obtained between elements of the convolution kernel and the values of input is calculated at each location and summed. This process is shown in FIGURE 2.7

In figure an Input matrix also known as input tensor is element-wise multiplied and summed up to the output also known as a feature map, figure b and c show that this process is repeated for each location and feature maps are obtained. Various paddings, mostly zero padding can be used to convolve boundary values of the input tensor with the convolution kernels. [20], [21]





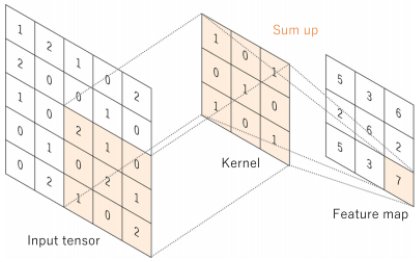


Figure 2.7: Application of Convolution Kernels over input tensors and feature maps

For CNNs containing more than one Conv layers feature maps of previous layer are dealt as input tensors of the next layer.

Other important terms include stride length, the distance between two locations on which kernels are applied is known as stride length, stride length is usually kept one. Moreover, kernel weights are shared and kept same for all the locations of input tensors which allows kernels to detect local patterns and increase model's efficiency.

Activation Functions

Output of convolution layers is then fed through non-linear activation functions before down sampling. Most of the modern CNNs use rectified linear unit (ReLU) function [19], [22]which can be stated simply in an equation as

F(x) = max (0, x)

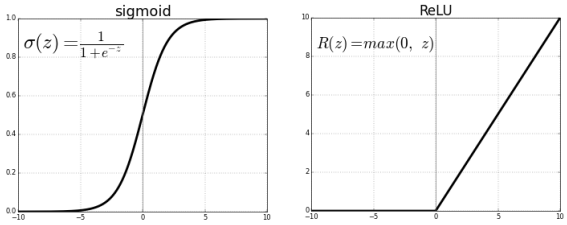


Figure 2.8: Most common activation functions sigmoid(left) and ReLU(right)

2.1.2.2 Pooling Layer

Several Down sampling techniques such as max pooling and global average pooling can be used after application of non-linear activation functions. In max pooling portions from input feature maps are obtained and maximum of each portion is kept while deleting all other values. Max pooling kernel sizes are specified before training. [Figure] shows the functioning of a max pooling layer. In a similar but more severe fashion global average pooling reduces the dimensions of each input feature to a 1 by 1 value by taking average of all values. This allows dramatic decrease of variable input tensors to one value. It is important to note that down sampling decreases the size of tensors therefore limiting the complexity of a convolutional neural network. [9]

2.1.2.3 Fully Connected/ Dense Layer

Output of non-linear activation or pooling layers is then converted to one dimensional array and connected to one or more fully connected flat (One Dimensional) layers. In these layers every input and output are connected through a learnable weight. Each connected layer is also passed through non-linear activation functions such as ReLU. The final flat layer typically has the same number of nodes as the number of classes the input data must be classified into.

Last layer's activation function is chosen very carefully and is usually different from previous fully connected layers. One of the most common used layers is SoftMax, which normalizes the results between 0 and 1 based on the probability of each class.

Parameters and Hyperparameters are summarized in the table below

Table 2.1 Parameters and Hyperparameters of Convolutional Neural Networks

	Parameters	Hyperparameters
Convolution layer	Kernels	Kernel size, number of kernels, stride, padding, activation function
Pooling layer	None	Pooling method, filter size, stride, padding
Fully connected layer	Weights	Number of weights, activation function
Others		Model architecture, optimizer, learning rate, loss function, mini-batch size, epochs, regularization, weight initialization, dataset splitting

Note that a parameter is a variable that is automatically optimized during the training process and a hyperparameter is a variable that needs to be set beforehand

CHAPTER 3: Methods and Materials

3.1 Experimental Paradigm and Data Specifications

Two open source data-sets [23], [24] were used together to train Convolution Neural Networks, both contained experiments commonly used for Brain Computer Interfaces namely
Word Generation (WG), Motor Imagery (Different tasks for left- and right-hand motor imagery (L- and R-MI)) and Mental Arithmetic (MA). Each experiment contained both resting state data and activity. In resting state data, the subject did not perform the task and relaxed and in Active trial the subject performed the mentioned tasks.
During the recording a montage of sources and detector probes were attached to the skin through a cap on the head of a subject. The source optodes emit near-infrared light which is reflected from the brain tissue and is received by the detector. This near Infra-red light is relatively left unabsorbed by the brain tissue and bone and its absorption chiefly depends on the amount of hemoglobin it encounters. 36 Channels were configured using adjacent sources and detectors in the data as shown in the figure (Dark Circles)

Fnirs Channels Used Quinary Classification can be seen in red in the following diagram

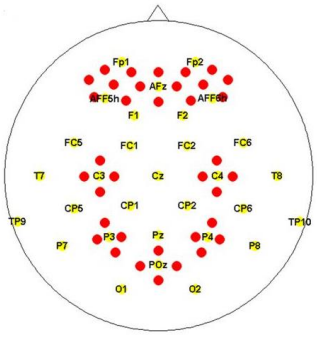


Figure 3.1: Locations of placed optodes in the experiment with reference to their EEG locations

Experimental Paradigm

Dataset A

The subjects sat on a comfortable armchair in front of a 50-in white screen. The distance between their heads and the screen was 1.6 m. They were asked not to move any part of the body during the data recording. The experiment consisted of three sessions of left- and right-hand MI and, MA and baseline tasks (taking a rest without any thought) each. Each session comprised a 1 min pre-experiment resting period, 20 repetitions of the given task and a 1 min post-experiment resting period. The task started with 2 s of a visual introduction of the task, followed by 10 s of a task period and resting period which was given randomly from 15 to 17 s. At the beginning and end of the task period, a short beep (250 ms) was played. All instructions were displayed on the white screen by a video projector. MI and MA tasks were performed in separate sessions but in alternating order [i.e., sessions 1, 3 and 5 for MI and sessions 2, 4, and 6 for MA].

Dataset B

The participants sat on a comfortable chair in front of a 24’ LCD monitor. The distance between the participants’ eyes and the monitor was approximately 1.2 m. They were instructed to keep their eyes on the monitor and refrain as much as possible from moving their body throughout the data recording. The experiment consisted of three sessions of n-back, DSR, and WG task each.

Only Word Generation task was selected from these three tasks to be used for training data for both binary and quinary classification since it contained temporally separated trials

Temporal Sequence of Tasks in both data-sets

Each trial started with a 2s instruction on a screen. The subject performed the instruction in the ten seconds after the initial two seconds. Task period always started with a short (250ms) beep and ended with a STOP sign on the screen which was displayed for 1 second. The task period was followed by 13-17 seconds of rest period before the next instruction was shown.

Figure [] shows the temporal sequence of all tasks.

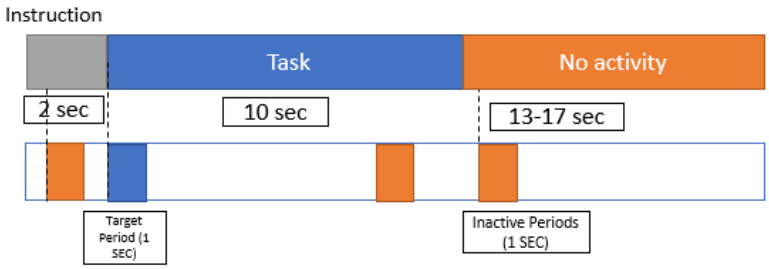


Figure 3.2: Three phases of trials and their division

Classes of Data

For the quinary classification the task trials were categorized into Right motor imagery (RMI), Left Motor Imagery (LMI), Mental Arithmetic (MA), and Word generation (WG). Both datasets were used in epoching of data. Our classifier needs to be able to detect the first second of active trials from remaining seconds of the trial. It also needs to classify the type of active trial it is from LMI, RMI, WG & MA.

Word Generation Task (WG)

Instructions in the initial two seconds either showed a fixation cross or a random letter from the alphabet. Subjects had to rest or generate words from the shown letter in the 10 seconds of task after which they rested 13-15 seconds. This task produced 780 trials of both rest and intentional rest.

Right Hand and Left-Hand motor imagery (RMI & LMI)

Subjects imagined moving their left or right hand depending on the instruction shown in the initial two seconds. This task provided 870 trials of both Left- and Right-Hand Motor Imagery.

Mental Arithmetic (MA)

In this task subjects were shown multiple arithmetic questions on screen which they had to solve mentally in the ten second task period. This task constituted of 870 Active trials and 870 intentional rest periods.

Resting Periods or Baseline (Rest)

In WG and MA tasks subjects were also explicitly directed half of the time to relax instead of performing the task. These intentional rests, and resting periods after the tasks will together be called as resting periods.

3.2 Methods

3.2.1 Preparation of data and Objectives

Data preparation is often used to describe the storage of data where each trial is stored with its class label so it can be used to train CNN's. It is best to divide the data into three different types with as much homogeneity as possible namely Training, Validation and Testing data. [9], [19]
Training data which composes of majority of the data is used to train CNN to recognize the features from different trails and improve its classification method. After training CNN apply the learned parameters to the validation data to determine its accuracy and to keep a check on whether the network is not overfitting or "memorizing data".[25] Testing data is used after the training and validation process to test the final accuracy of models over completely unfamiliar trials of data. Testing accuracy reveals the real accuracy and thus is the best measure of goodness of a model. It is often used to choose the best model from a bunch of trained models. Other efficiency constraints along with accuracy of classification include the average time a model take to classify a presented trial. This is also measured over the testing data. Each trial's duration may be measured separately, and statistical analysis is applied over it.

3.2.2 Preprocessing Before Training CNN models

One of the best features of CNN is that it does not need to be fed with explicitly extracted features, since convolution layers extract useful features automatically in a hierarchal manner. This avoids bias of explicit feature extraction which may ignore important features.
Several methods can still improve efficiency of CNNs significantly e.g. Down sampling of data, resizing all images to same sizes and Data normalization into a [0-1] range are among the most popular and useful methods.

3.2.3 CNN and it's training

To accommodate the need of speed of a classifier while keeping it complex enough so it can accurately classify enough classes, A compromise between complexity and speed of classification is unavoidable. In our case that is using Convolutional Neural Networks for BCI classification,

speed and accuracy are the ultimate priorities. It is normal for a subject to expect movement or his task being executed as quickly as he or she decides to perform it. Therefore, we have kept the classification as fast and accurate as possible. Convolutional Neural Networks as seen in the literature review can get increasingly large and require a lot of computational power and time to but if they are kept from getting complex, they can still be useful in classifying slow changing signals such as fnirs.

A CNN has two types of characteristics known as Parameters and Hyperparameters that need to be adjusted according to training requirements. A parameter's values are adjusted during training automatically to obtain the best possible results e.g. weights of fully connected layers, values of convolution kernels etc. whereas hyperparameters make up the structure of network thus hyperparameters are selected either manually before starting training of the network or a multiple training session are iterated over specified range of hyperparameters. Some very simple examples of hyperparameters include Number of each layers, choice of back propagation and optimization techniques also other choices such as use of different pooling techniques and activation functions.

3.2.3.1 Different Hyperparameters and their uses

Hyperparameters can be classified according the layers which they define, they are briefly explained before under the headings of the layers they define

a. Convolution Layer Hyperparameters

In order to define a convolution layers in CNN we need to know how many layers to use and in which order. We also need to know which activation functions follow each layer e.g. activation functions such as RelU, step, linear or sigmoid. ReLu is used mostly since it resembles actual activation of neurons. [9]

Number and sizes of convolutional layers define the complexity of a network, time efficiency chiefly depends on these since more complexity gives rise to more processing time. After the total number and sequence of convolution layers is decided. Size of each convolutional layer's kernels is decided. Each kernel is a small tensor which has adjustable values that adjust during training, its initial values are usually temporary.

Other convolution hyperparameters include stride length which determines the unit number of pixels over which a convolution kernel moves on an image/tensor. It is mostly kept one, increasing it is equivalent to down sampling an image before applying convolution by the stride length factor. To apply convolution function on edges of input tensors we need to extrapolate layers of pixels around the border this is called padding, type of paddings must be chosen before applying convolution most common type of padding used is zero padding in which a boundary of zeros is applied at the edges of input tensor.

b. Pooling Layers

Pooling layers are inserted in between convolution layers to down sample feature maps, several types of pooling include global average pooling, SoftMax pooling and Max pooling. Max pooling which is used most frequently determines the greatest value in a fixed sized kernel from the feature maps in other words it selects the maximum value from a patch of input and rejects the rest. Global Average pooling just outputs the calculated average of whole feature map. The advantages of applying global average pooling are that it reduces the number of learnable parameters and enables the CNN to accept inputs of variable size. Hyperparameters that need to be determined here are size of the pooling patches and type of pooling.

c. Hyperparameters of Fully Connected Layer

Before feeding feature maps to dense layers they are flattened that is converted to vectors, usually there are no different types of flattening the data is usually reduced in dimension also known as flattening

Fully connected layers resemble convolution layers in the sense that they have fixed sizes and activation functions can be used after each of them. But they do not extract features instead train their weights over already extracted features in convolutional layers. Hyperparameters here include number of nodes/weights per each dense layer and type of activation function.

Last activation function before classification is usually different from other activation functions since it must classify the outputs into clear different classes. The choice of last activation function depends on the types of classification tasks. For multiple classes problems a SoftMax function is used which gives result in terms of probabilities per each class. Probabilities as usual are normalized with 1(ONE) being a 100% probability.

Table 3.1 Last Layer Activation choices

Task	Last layer activation function
Binary classification	Sigmoid
Multiclass single-class classification	Softmax
Multiclass multiclass classification	Sigmoid
Regression to continuous values	Identity

Table 3.2 Summary table of parameters and Hyperparameters

	Parameters	Hyperparameters
Convolution layer	Kernels	Kernel size, number of kernels, stride, padding, activation function
Pooling layer	None	Pooling method, filter size, stride, padding
Fully connected layer	Weights	Number of weights, activation function
Others		Model architecture, optimizer, learning rate, loss function, mini-batch size, epochs, regularization, weight initialization, dataset splitting

Note that a parameter is a variable that is automatically optimized during the training process and a hyperparameter is a variable that needs to be set beforehand

3.3 Methodology

3.3.1 Preparation of Data

One second periods from the required channels were cut from the start of active trials (BCI tasks) and three different non-active one second trials were randomly cut from each resting state time periods after the task and from the remaining task period (9 seconds) as shown in the figure using **Matlab** script

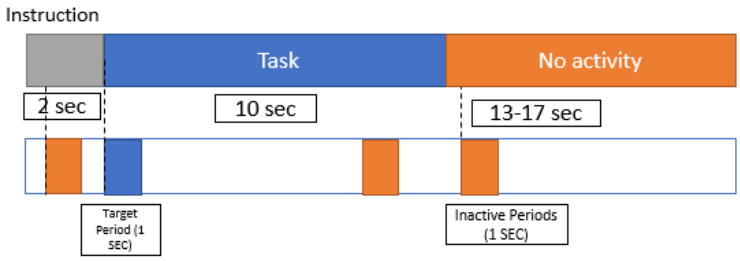


Figure 3.2: The Phases of Trials

3.3.2 Conversion of Trials to 2D Tensors and their storage

Data is converted to two dimensional tensors before training models, each tensor contains one second of raw fNIRS data from various channels. Each row of the image contains oxygenated or deoxygenated data from different channels. Size of each image is Number of channels by Sampling Frequency also using **Matlab**. For convolutional Neural Networks the sequence of channels (rows) and data values (columns) does not affect the classification accuracy if they are kept same for all images[9].

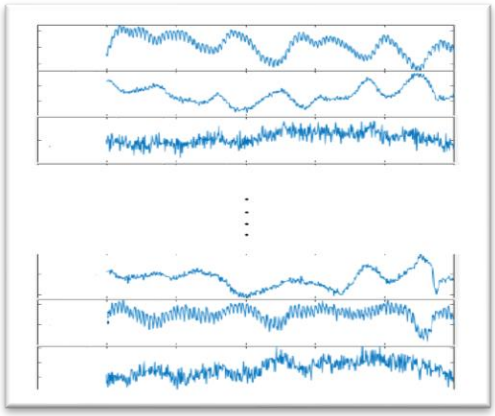


Figure 3.3: Illustration of an Image [72 rows and 10 columns (1second)]

Table below shows a summary of Number of trials per each class, before making them all equal by replicating the activity periods

Table 3.3: Summary of trials of each class for quinary classification

Task		Label	Total Trials
Word Generation	WG	4	780
Right Hand Motor Imagery	RMI	3	870
Left Hand Motor Imagery	LMI	2	870
Mental Arithmetic	MA	1	870
Baseline / Rest	Rest	0	15,120

3.3.3 CNN and its training

A simple structure of CNN was chosen to classify trials for BCI, it constitutes of convolution and pooling layers appearing periodically in a sequence before being flattened and fed to dense layers for classification.

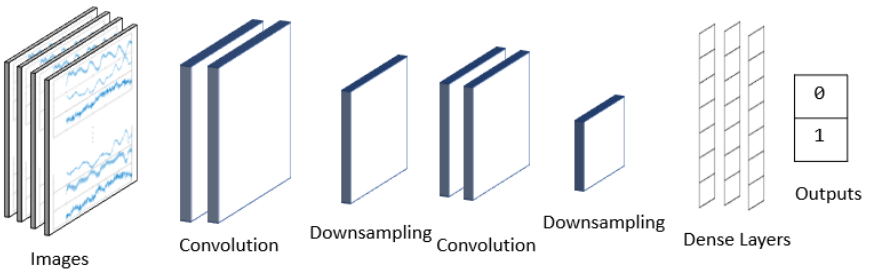


Figure 3.4: Pattern Diagram of used CNNs

This figure shows a CNN where various hyperparameters are chosen separately to train numerous models, following table summarizes the range of hyperparameters chosen for all models

Table 3.4 Hyperparameter range for comparing accuracies

Hyperparameter	Min	Max	Increment
Number of Convolution and Pooling layers	1	3	+1
Fully Connected Layers	0	2	+1
Layers Size	10	20	+5
Batch Sizes	10	20	+5

Table 3.5 Chosen Optimization methods and other procedure hyperparameters

Loss	Sparse Categorical Cross Entropy
Optimization	Adam Optimization
Activation Functions	RELU after each layer except output layer
Model Callbacks	Tensor Board
Stride Length	1
Convolution Kernel Size	3x3
Max Pooling Kernel Size	2x2

3.3.4 Platforms/Tools used for Methodology

A simple laptop **Core-M3** was used to train and analyze data and to achieve greater than 95% accuracy and classification time of less than 2 milli-seconds per trial.

Table 3.6 Specifications of system used for testing data

Processor:	Intel(R) Core(TM) m3-7Y30 CPU @ 1.00GHz 1.61 GHz
Installed memory (RAM):	8.00 GB
System type:	64-bit Operating System, x64-based processor
Pen and Touch:	Touch Support with 10 Touch Points

Additional Solid-State Drive (160GB) was installed for higher reading and writing speed of data
All the preprocessing including epoching and sorting of training data was performed using **Matlab** since the data files were already available in 'mat' formats and experimental information was stored in Matlab structures along with it.
After epoching all classes, they were saved in separate folders and trained using **TensorFlow**, this shift from Matlab to python was necessary because of higher speed of python specially in training deep neural networks.

Tensorboard was used to keep track of validation and training process to ensure the models were not overfitting and moving towards more accuracy with each iteration.

Chapter 4: Results and Discussion

4.1 Performance of selected hyperparameter ranges

Following table summarizes the results of models with top five performances in terms of classification times and accuracies, Number of kernels for conv layers and weights for dense layers were kept same

Table 4.1 Summary of results with different hyperparameters

Hyperparameters				**Test Accuracy (Percentage)**		**Average Time To classify one trial (Milli-seconds)**
Convolution Layers (3x3)		**Pooling Layers (2x2); Max Pooling**	**Dense Layers [0-2]**			
Kernels	**Number**	**Number**	**Number**	**Validation**	**Test**	
20	2	2	1	100	96.448	1.914
15	2	2	1	99.6	94.924	1.889
20	2	2	0	99.6	89.452	1.795
15	2	2	0	99.4	87.793	1.784
10	2	2	0	99.8	84.675	1.696

The table shows the trends of classification times, accuracies and their dependence on the complexity of our networks. With an increase in number of convolution layers accuracy increases significantly before saturating at two layers. Similarly complicating the network's dense layers is only useful until a dense single layer. Since every dense layer causes an increase in the classification time of the network.

4.2 Effects on performances with different hyperparameters

4.2.1 Number of Convolution Layers

Increasing convolution layers while using no dense layers and 15 kernels

Table 4.2 Classification times and accuracies with different number of convolution layers

Number of Layers	Classification Time (ms)	Testing Accuracies (%)
1	1.532	77.343
2	1.784	87.793
3	2.051	87.801

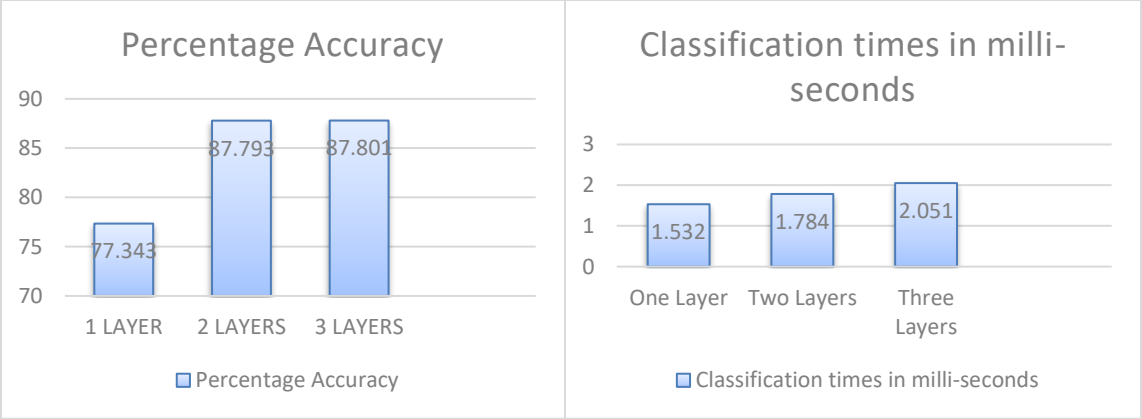


Figure 4.1 : Classification times and accuracies with different number of convolution layers

4.2.2 Number of Dense Layers

Increasing Dense Layers while using 20 kernels and 2 convolution layers

Table 4.3 Classification times and accuracies with different number of dense layers

Number of Layers	Classification Time (ms)	Testing Accuracies (%)
0	1.795	89.452
1	1.914	96.448
2	2.203	96.551

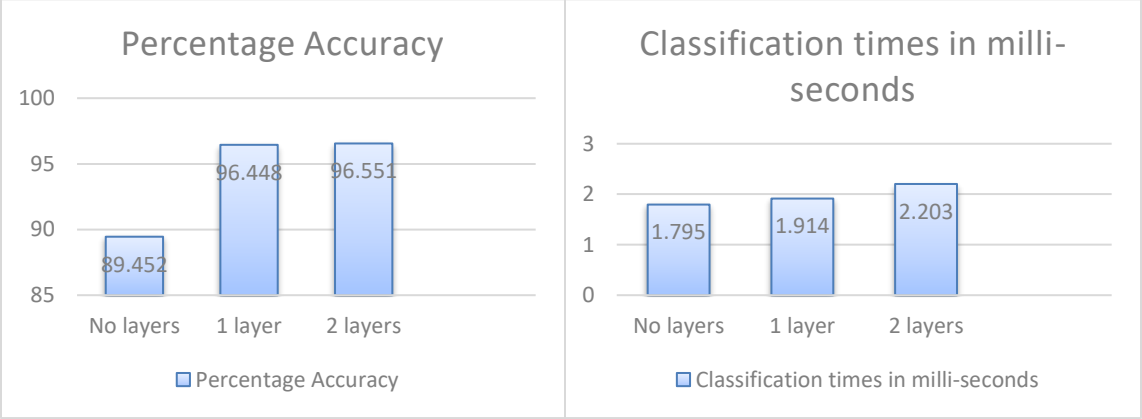


Figure 4.2 Classification times and accuracies with different number of dense layers

4.2.3 Number of Kernels and Nodes

Increasing the number of kernels from 10 to 25 of the convolution layers while keeping number of convolution and dense layers same i.e. two convolution layers with one dense layer result in following performance of networks

Table 4.4 Classification times and accuracies with different number of kernels and weights

Number of kernels/nodes	Classification Time (ms)	Testing Accuracies (%)
10	1.852	93.194
15	1.889	94.924
20	1.914	96.448

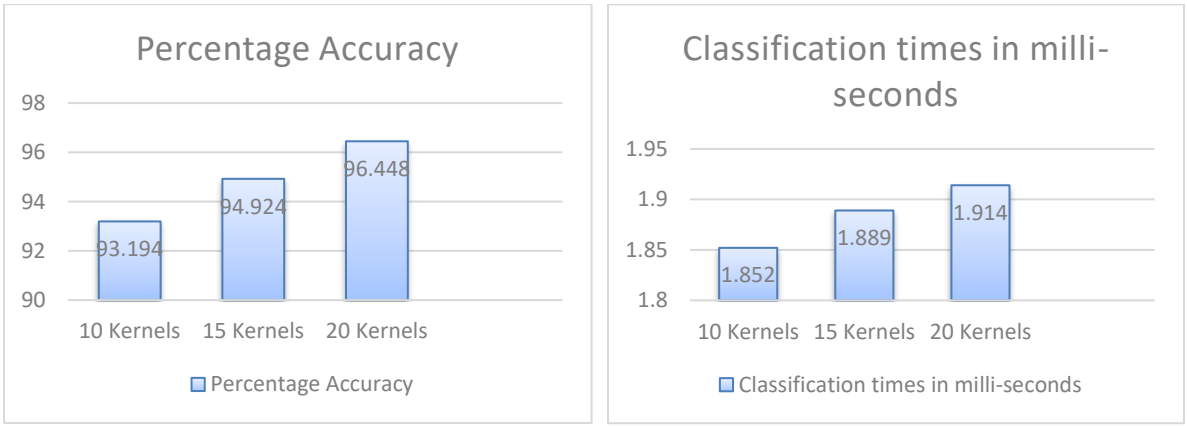


Figure 4.3 Classification times and accuracies with different number of kernels and weights

4.3 Different Accuraccies and Losses

4.3.1 Epoch Accuracy and Loss

Epoch accuracy plots the accuracy of training model over the training data, each subsequent epoch is a better attempt at finding local maxima, that are values of parameters such as weights of dense layers and element-values of convolutional kernels that yield better accuracy results. These are not the final measure of performance of a model since they only show how a model is performing on the "seen" training data. Epoch loss is like epoch accuracy and signifies the error rate per epoch. Following snippets show progress of our models reaching 80 to 100% testing accuracies around 30th epochs.

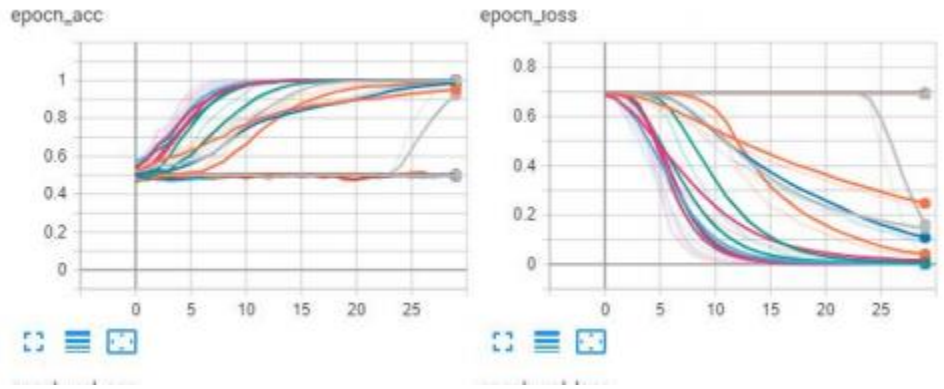


Figure 4.4 : Progress of epoch acccuracies and losses

4.3.2 Epoch Validation accuracy (epoch_val_acc) and epoch validation loss (epoch_val_loss)

A chosen percentage from data is kept separate from training data during training process this data is known as validation data and is used to confirm the training accuracies on unused data, this data is different from testing and training data, it is shown to the model after every epoch, although it usually does not directly affect the parameters of a model but it is very helpful in predicting over-fitting. Testing data on the other hand is kept disused during the whole training process. Following snippets show how Tensorboard displays the progress of validation errors of our models

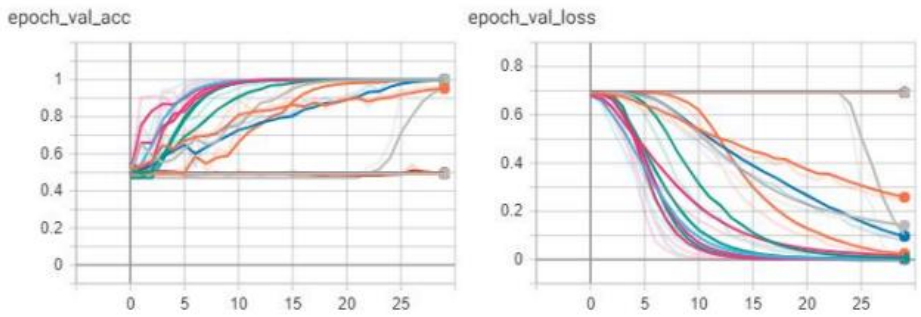


Figure 4.5: Progress of models with respect to validation accurcies and validation losses

Ideal validation curves show similar trends to epoch accuracies and losses, presence of abrupt rises or falls different from epoch accuracies mean the model is not performing same on training and validation data therefore there are chances of overfitting.

4.3.3 Testing Accuracy

Test data is only used after the parameters of models are fully trained i.e. this data is kept hidden from the model during all its epochs of training. Testing accuracy is the final measure of a model's accuracy performance.[21], [26]

A summary of predictions along with true labels over the testing data is shown in the following bar graph

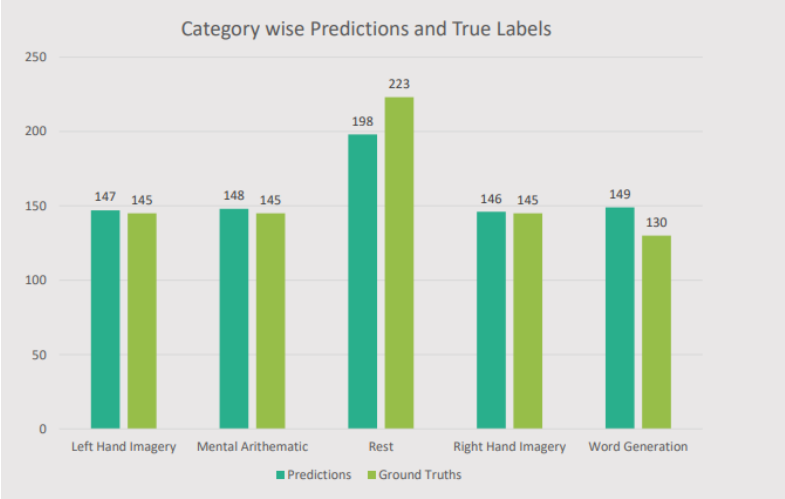


Figure 4.6: Predictions (Dark) vs Ground Truths (Light)

An overview of all testing predictions can be seen through the following image, in this image each pixel denotes a trial, each category has its own sseparate color as shown in the legend. Oddities in the image show wrong predictions **(p<0.05)**

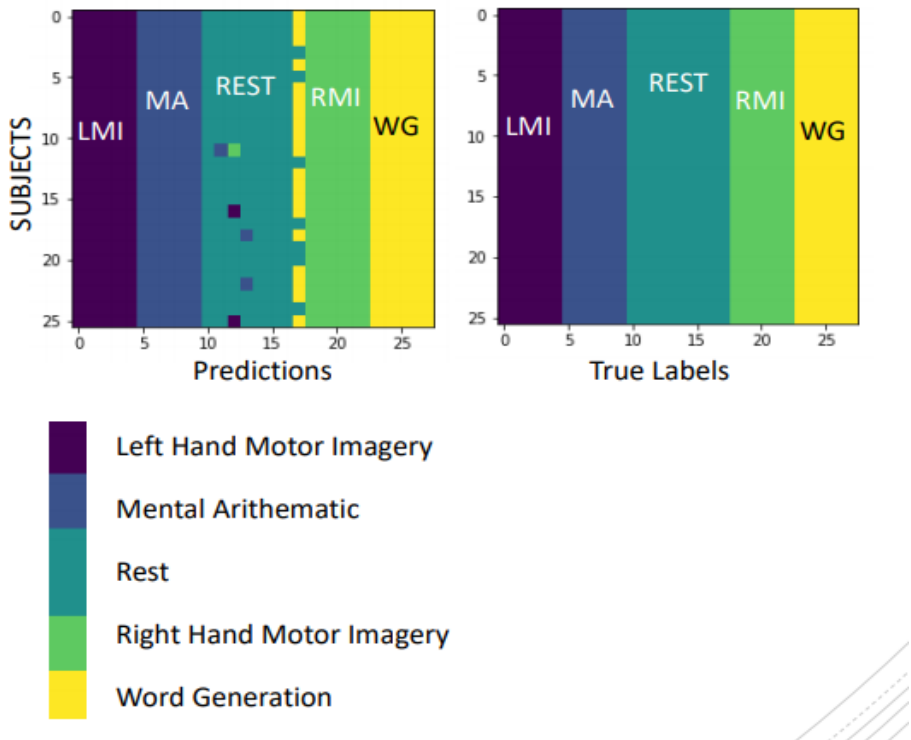


Figure 4.7: Test data analyses

Summary and Conclusion

Deep Lerning is an emerging field of Artificial Intelligence[27] and it has numerous applications in brain computer interfaces, chiefly because of its inherent ability to extract useful features. Brain Computer Interfaces require high accuracy and fast classification times which can limit how complex a Network can be. A useful model should be able to classify small windows of data accuratly and as quickly as possible on simple processors that can work with BCIs. Our convolutional neural network classifies one second windows of fNIRS signals down sampled to 10Hz only with excellent accuracy i.e. as much as 96 percent. The preparation is very simple and includes converting data into simple images [rows into coloumns]. A lot of useful information is obtained by training various models over various configurations of data such as effects of changing hyperparameters, complexity, number of channels and number of categories that the windows can be ssorted into. These characteristics makes CNNs an effective and fast technique for practicable translation of data to commands for machines.

APPENDIX

5.1 MATLAB CODES FOR FNIRS DATA PREPARATION

USED FUNCTIONS

Moves multiple files from pickup to dropoff folder

```
function move_rand(pickup,dropoff,names)
cd(pickup)
for i=1:numel(names)
x = names{i};
movefile(x,dropoff)
end
end
```

For Visual inspection and saving specific portions of fnirs data experiment-B

```
%% this function plots required data
for visual inspection and returns the
data to workspace

%% Description
% Inputs
% cnt_dsr : data variable oxy and deoxy
% o : mode selection 0-EEG (-1)->ALL
% 1-(oxygenated) 2-(Deoxygenated) 3-
(Both)
% chan: vector containing channels to
be plotted
% time vector contains two elements
starting and ending time in minutes
%
% Default Publish : DATA =
fnirsplot(cnt_dsr,3,[1 2],[0,5])

%% Code
function crop =
fnirsplot(cnt_dsr,o,chan,time) %
(file,mode,channels,[start end])

if o~=0

 XX = 0:1/600:numel(cnt_dsr.oxy.x)/600;
%FOR MINUTES [10Hz*60Sec]
 %XX = 0:1/10:14800/10; %FOR SECONDS
sampling rate [10Hz]
 s  =    time(1);
 e  =    time(2);
 %  figure;
 if (s==0)
     s=1/600;
 end
 crop.time = time;
 crop.x(1,:) = XX(s*600:e*600); %
minutes

end


 %% EEG
 if o==0
     XX = 0:1/12000:394930/12000; %FOR
MINUTES [10Hz*60Sec]
 %XX = 0:1/10:14800/10; %FOR SECONDS
sampling rate [10Hz]
 s  =    time(1);
 e  =    time(2);
 %  figure;
 if (s==0)
     s=1/12000;
 end
 crop.time = time;
 crop.x(1,:) = XX(s*12000:e*12000); %
minutes
      figure;
       for i=1:numel(chan)
```

```
            crop.y(i,:) =
cnt_dsr.x((s*12000:e*12000),chan(i));
%data
            crop.type = 'MINUTES';
            crop.channel(i) = chan(i);
            crop.eegchannel(i) =
cnt_dsr.clab(chan(i));

        subplot(numel(chan),1,i)


plot(crop.x,cnt_dsr.x((s*12000:e*12000)
,chan(i)))
         str = 'MINUTES';
         xlabel(str)
         ylabel(crop.eegchannel(i))


        end

 end

 %% OXY
 if o==1
        figure;
       for i=1:numel(chan)
        crop.y(i,:) =
cnt_dsr.oxy.x((s*600:e*600),chan(i));
%data
        crop.type = 'OXY - MINUTES';
        crop.channel(i) = chan(i);
        crop.eegchannel(i) =
cnt_dsr.oxy.clab(chan(i));

        subplot(numel(chan),1,i)


plot(crop.x,cnt_dsr.oxy.x((s*600:e*600)
,chan(i)))
         str = 'OXY - MINUTES';
         xlabel(str)
         ylabel(num2str(chan(i)))


       end
 end
 %% DEOXY
 if o==2
%         figure;
       for i=1:numel(chan)

        crop.y(i,:) =
cnt_dsr.deoxy.x((s*600:e*600),chan(i));
%data
        crop.type = 'DEOXY - MINUTES';
        crop.channel(i) = chan(i);
        crop.eegchannel(i) =
cnt_dsr.deoxy.clab(chan(i));


%         subplot(numel(chan),1,i)

%
plot(XX(s*600:e*600),cnt_dsr.deoxy.x((s
*600:e*600),chan(i)))
%           str = 'DEOXY - MINUTES';
%           xlabel(str)
%           ylabel(num2str(chan(i)))
        end
 end
 %% BOTH
 if o==3

       for i=1:numel(chan)


        crop.y(2*i,:) =
cnt_dsr.deoxy.x((s*600:e*600),chan(i));
        crop.y(2*i-1,:) =
cnt_dsr.oxy.x((s*600:e*600),chan(i));
        crop.channel(i) = chan(i);

%       figure;
%         subplot(2,1,2)
%         plot(crop.x,crop.y(2*i,:));
%         str = ['DEOXY - MINUTES ',
cnt_dsr.deoxy.clab(chan(i))]; %
displaying eeg location
%         ylabel(num2str(chan(i)))
%         xlabel(str)
%
%         subplot(2,1,1)
%         plot(crop.x,crop.y(2*i-1,:))
%         str = ['OXY - MINUTES',
cnt_dsr.oxy.clab(chan(i))]; %
displaying eeg location
%         xlabel(str)
%         ylabel(num2str(chan(i))) %
channel number
%
       end
 end
 %% Moving Average
 if o==4

     for i = 1:numel(chan)
     figure;
     %%
        t(:,i) =
mov(cnt_dsr.x(:,chan(i)),300);
     %%
        crop.y(2*i,:) =
cnt_dsr.x((s*600:e*600),chan(i));
        crop.y(2*i-1,:) =
t((s*600:e*600),i);
        crop.eegchannel(i) =
cnt_dsr.clab(chan(i));
        crop.type = 'Moving average';

     %%
        subplot(2,1,2)

plot(XX(s*600:e*600),crop.y(2*i,:))
                xlabel([cnt_dsr.xInfo
cnt_dsr.clab(chan(i))])
```

```
        subplot(2,1,1)

plot(XX(s*600:e*600),crop.y(2*i-1,:))
                xlabel(crop.type)
     end

 end

 if o==5
     figure;
        cnt_dsr =
downsample(cnt_dsr,2);
        div =
diff(diff(cnt_dsr(s*300:e*300)));
        plot(div)

figure;plot(cnt_dsr(s*300:e*300))
        movd = mov(div,5);
        movd = movd.*movd;
        movd = movd.*movd;
        figure;plot(movd)
 end

%% ALL
if o==-1
    for i=1:numel(chan)
       %%
       figure;
        crop.y(2*i,:) =
cnt_dsr.deoxy.x((s*600:e*600),chan(i));
        crop.y(2*i-1,:) =
cnt_dsr.oxy.x((s*600:e*600),chan(i));
        crop.channel(i) = chan(i);
       %%
        subplot(3,1,2)
        plot(crop.x,crop.y(2*i,:));
        str = ['DEOXY - MINUTES ',
cnt_dsr.deoxy.clab(chan(i))]; %
displaying eeg location
        ylabel(num2str(chan(i)))
        xlabel(str)

        subplot(3,1,1)

        plot(crop.x,crop.y(2*i-1,:))
        str = ['OXY - MINUTES',
cnt_dsr.oxy.clab(chan(i))]; %
displaying eeg location
        xlabel(str)
        ylabel(num2str(chan(i))) %
channel number

        subplot(313)

        XX = 0:1/12000:394930/12000;
%FOR MINUTES [10Hz*60Sec]
        %XX = 0:1/200:14800/10; %FOR
SECONDS sampling rate [10Hz]
        s  =    time(1);
        e  =    time(2);
        %  figure;
        if (s==0)
         s=1/12000;
        end
        crop.time = time;
        crop.x(1,:) =
XX(s*12000:e*12000); % minutes



plot(crop.x,cnt_dsr.x((s*12000:e*12000)
,chan(i)))
         str = 'MINUTES';
         xlabel(str)
         ylabel(num2str(chan(i)))

end

end
end
```

DEVELOPED SCRIPTS

READING WINDOWS OF DATA FROM EXPERIMENT-A FILES

```
for wl = 9
    X =
'C:\Users\ahmad\Desktop\MODELS\NIRS_01-
29';
    mat = dir(X);
    Y =
'C:\Users\ahmad\Desktop\MODELS\PROCESSE
D\tenx72';
    for t=3:31
        disp(['reading subject number :
',num2str(t-2)])
        p = [X ,'/', mat(t).name];
        max1 = dir([p , '/*.mat']);
% SUBJECT 16 LACKS ARTIFACTS
        if (t~=16)

load(max1(1).name);load(max1(3).name);l
oad(max1(5).name)
        else
```

```
load(max1(1).name);load(max1(2).name);l
oad(max1(3).name)
        end
% c- data % -i lmi/rmi % -a bls/ma % m-
details
        ci = [cnt(1,1) , cnt(1,3) ,
cnt(1,5)];
        ca = [cnt(1,2) , cnt(1,4) ,
cnt(1,6)];
        mi = [mrk(1,1) , mrk(1,3) ,
mrk(1,5)];
        ma = [mrk(1,2) , mrk(1,4) ,
mrk(1,6)];
% 3 I-'s and 3 A-'s
        for k=1:3
% 20 X 2 X 3 X 29 X 1  TRIALS
            for i=1:20
% trial_end = ending time of each trial
% last element for last trial (20)
                if i~=20
                    trial_end_i =
round(((mi{1,k}.time(i+1))/100));
                    trial_end_a =
round(((ma{1,k}.time(i+1))/100));;
                else
                    trial_end_i =
size(ci{1,k}.x',2);
                    trial_end_a =
size(ca{1,k}.x',2);
                end

                s=
round(((mi{1,k}.time(i)+2000)/100));
                e=
round(((mi{1,k}.time(i)+2000)/100)+wl);
                sa=
round(((ma{1,k}.time(i)+2000)/100));
                ea=
round(((ma{1,k}.time(i)+2000)/100)+wl);

                if
mi{1,k}.event.desc(i) ==1
                    lmi =
matcut(ci{1,k}.x',s,e) ;
                    rlmi =
matcut(ci{1,k}.x',[s-20,e+1],[s-
1,trial_end_i]);

                    save([Y,'\lm\lmi',
num2str(t-2), 'wl_' ,num2str(wl),'k_'
,num2str(k),'trial_',
num2str(i)],'lmi');

save([Y,'\r_lm\rlmi', num2str(t-2),
'wl_' ,num2str(wl),'k_'
,num2str(k),'trial_',
num2str(i)],'rlmi');


                elseif
mi{1,k}.event.desc(i) ==2
                    rmi =
matcut(ci{1,k}.x',s,e) ;
                    rrmi =
matcut(ci{1,k}.x',[s-20,e+1],[s-
1,trial_end_i]);


                    save([Y,'\rm\rmi',
num2str(t-2), 'wl_' ,num2str(wl),'k_'
,num2str(k),'trial_',
num2str(i)],'rmi');

save([Y,'\r_rm\rrmi', num2str(t-2),
'wl_' ,num2str(wl),'k_'
,num2str(k),'trial_',
num2str(i)],'rrmi');


                end


                if
ma{1,k}.event.desc(i) ==1

                    mas =
matcut(ca{1,k}.x',sa,ea) ;
                    rmas =
matcut(ca{1,k}.x',[sa-20,ea+1],[sa-
1,trial_end_a]);

                    save([Y,'\ma\mas',
num2str(t-2), 'wl_' ,num2str(wl),'k_'
,num2str(k),'trial_',
num2str(i)],'mas');

save([Y,'\r_ma\rmas', num2str(t-2),
'wl_' ,num2str(wl),'k_'
,num2str(k),'trial_',
num2str(i)],'rmas');


                elseif
ma{1,k}.event.desc(i) ==2

                    blsx =
matcut(ca{1,k}.x',sa,ea) ;
                    rblsx =
matcut(ca{1,k}.x',[sa-20,ea+1],[sa-
1,trial_end_a]);


save([Y,'\ma_rest\blsx', num2str(t-2),
'wl_' ,num2str(wl),'k_'
,num2str(k),'trial_',
num2str(i)],'blsx');

save([Y,'\r_ma_rest\rblsx', num2str(t-
2), 'wl_' ,num2str(wl),'k_'
,num2str(k),'trial_',
num2str(i)],'rblsx');

                end
            end
```

```
        end
    end
```

READING WINDOWS OF DATA FROM EXPERIMENT-B FILES

```
for wl = 9
    path =
'C:\Users\ahmad\Desktop\MODELS\PROCESSE
D\garbs';

    mkdir(path,'rst')
    mkdir(path,'rrst')
    mkdir(path,'wg')
    mkdir(path,'rwg')

    for t=1:26
        disp(['reading subject number :
',num2str(t)])
        X =
['C:\Users\ahmad\Desktop\MODELS\RAW\VP0
0',num2str(t),'-NIRS'];
        cd=X;
        load('cnt_wg.mat');
        load('mnt_wg.mat');
        load('mrk_wg.mat');


        data_x =
fnirsplot(cnt_wg,3,[1:36],[0 33]);
        j=1;
        k=1;



%     all_trials = randperm(60);
%
save(['C:\Users\ahmad\Desktop\MODELS\re
sults\100\randomization',num2str(t)],al
l_trials)
%     training = all_trials(1:54);
%     testing = all_trials(55:60);

        for i=1:60

            s=
round(((mrk_wg.time(i)+2000)/100));
            e=
round(((mrk_wg.time(i)+2000)/100)+wl);
            if i~=60
                trial_end =
round(((mrk_wg.time(i+1))/100));
            else
                trial_end =
size(data_x.y,2);
            end
            if mrk_wg.y(1,i)==1 &&
mrk_wg.y(2,i)==0

                wg =
matcut(data_x.y,s,e);
                rwg =
matcut(data_x.y,[s-20,e+1],[s-
1,trial_end]); %this line

                save([path,'\wg\','WG',
num2str(t), '_' ,num2str(wl),'_'
,num2str(j)],'wg');

save([path,'\rwg\','RWG', num2str(t),
'_' ,num2str(wl),'_'
,num2str(j)],'rwg');

                j=j+1;

            end

            if mrk_wg.y(1,i)==0 &&
mrk_wg.y(2,i)==1
                    rst =
matcut(data_x.y,s,e);
                    rrst =
matcut(data_x.y,[s-20,e+1],[s-
1,trial_end]); %this line


save([path,'\rst\','RST', num2str(t),
'_' ,num2str(wl),'_'
,num2str(k)],'rst');

save([path,'\rrst\','RRST', num2str(t),
'_' ,num2str(wl),'_'
,num2str(k)],'rrst');

                    k=k+1;

            end

        end

    end
clear all
end
```

RANDOMLY DIVIDING DATA INTO TRAINING AND TESTING

```
pickup = 'D:\onex72\10\rnds';
dropoff =
'D:\onex72\10_test\subject_0';
t = dir(pickup);
% for u=1:29
%     cd(dropoff)
%     mkdir(['subject_0',num2str(u)])
% end

for i=1:29 % 29 SUBJECTS
    dropof = [dropoff,num2str(i)];
%% TASKS_29
    d_1 =
dir(['D:\onex72\10\rnds\lmi\*lmi',num2s
tr(i),'wl*.mat']);
    d_2 =
dir(['D:\onex72\10\rnds\ma\*mas',num2st
r(i),'wl*.mat']);
    d_4 =
dir(['D:\onex72\10\rnds\rmi\*rmi',num2s
tr(i),'wl*.mat']);

%% REST_29

    d_3blsx =
dir(['D:\onex72\10\rnds\rest\blsx',num2
str(i),'wl*.mat']); %30
    d_3rblsx =
dir(['D:\onex72\10\rnds\rest\rblsx',num
2str(i),'wl*.mat']); %300
    d_3rrmi =
dir(['D:\onex72\10\rnds\rest\rrmi',num2
str(i),'wl*.mat']);
    d_3rlmi =
dir(['D:\onex72\10\rnds\rest\rlmi',num2
str(i),'wl*.mat']);
    d_3rmas =
dir(['D:\onex72\10\rnds\rest\rmas',num2
str(i),'wl*.mat']);

    if i<=26
%% RESTS_26
        d_3rst =
dir(['D:\onex72\10\rnds\rest\RST',num2s
tr(i),'_99*.mat']); %30
        d_3rrst=
dir(['D:\onex72\10\rnds\rest\RRST',num2
str(i),'_99*.mat']);
        d_3rwg =
dir(['D:\onex72\10\rnds\rest\RWG',num2s
tr(i),'_99*.mat']);

%% TASK_26
        d_5 =
dir(['D:\onex72\10\rnds\wg\*WG',num2str
(i),'_99*.mat']);
    end
```

```
%% Randomly obtaining 5 of each task
and 8 types of rest for eaach subject
        k=1;
        for j = randperm(30,5)
        d1{k} = d_1(j).name;
        k=k+1;
        end

        k=1;
        for j = randperm(30,5)
        d2{k} = d_2(j).name;
        k=k+1;
        end

        k=1;
        for j = randperm(30,5)
        d5{k} = d_5(j).name;
        k=k+1;
        end

        k=1;
        for j = randperm(30,5)
        d4{k} = d_4(j).name;
        k=k+1;
        end

        d3{1} =
d_3blsx(randperm(30,1)).name;
        d3{2} =
d_3rblsx(randperm(300,1)).name;
        d3{3} =
d_3rlmi(randperm(300,1)).name;
        d3{4} =
d_3rmas(randperm(300,1)).name;
        d3{5} =
d_3rrmi(randperm(300,1)).name;

        if i <=26
        d3{6} =
d_3rst(randperm(30,1)).name;
        d3{7} =
d_3rrst(randperm(300,1)).name;
        d3{8} =
d_3rwg(randperm(300,1)).name;
        end
    %function
move_rand(pickup,dropoff,names)
        cd(dropof)

        mkdir('lmi')
        mkdir('ma')
        mkdir('rest')
        mkdir('rmi')
        if i<=26
            mkdir('wg')
```

```
move_rand([pickup,'\wg'],[dropof,'\wg']
,d5)
        end


move_rand([pickup,'\lmi'],[dropof,'\lmi
'],d1)

move_rand([pickup,'\ma'],[dropof,'\ma']
,d2)
```

```
move_rand([pickup,'\rmi'],[dropof,'\rmi
'],d4)

move_rand([pickup,'\rest'],[dropof,'\re
st'],d3)



       clear d3
end
```

5.2 PYTHON CODES

READING DATA INTO PYTHON VARIABLES

```
@author: ahmad
"""
import os
import scipy.io
#import pickle
import random

dirx = 'D:/onex72'
cats = ["lmi","ma","rest","rmi","wg"]
#D = os.listdir(dirx)
tt_data = []
D = ['10']

## Function To create data from a folders 40-45-50-55-60

def create_t_data():
    for wls in D: # wls is window length folder names 40-45 ....
        dx= os.path.join(dirx,wls,'rnds') # '40/subfolder'
        for categ in cats: # categories e.g. word and base
            path= os.path.join(dx,categ) #files in base/word
            class_num = cats.index(categ)
            for img in os.listdir(path):
                try:
                    new_array = scipy.io.loadmat(os.path.join(path,img))


                    if (class_num ==0):
                        for x in range(15):
                            tt_data.append([new_array['lmi'],0])

                    if (class_num ==1):
                        for x in range(15):
                            tt_data.append([new_array['mas'],1])

                    if (class_num ==2):
                        try:
                            tt_data.append([new_array['baseline'],2])
                        except:
                            try:
                                tt_data.append([new_array['blsx'],2])
                            except Exception:
                                   print('ERROR IN RESTS')

                    if (class_num ==3):
```

```
                        for x in range(15):
                            tt_data.append([new_array['rmi'],3])
                    if (class_num ==4):
                        for x in range(20):
                            tt_data.append([new_array['wg'],4])



                except Exception:
                    print('Error Reading files')

create_t_data()


# cutting to parts and mixing trial wise
dataRS = tt_data;

random.shuffle(dataRS)
# seperating
features = []
labels = []



for feature,label in dataRS:
        features.append(feature)
        labels.append(label)
```

Preparing, training, and saving models

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation , Flatten, Conv2D,
MaxPooling2D
from tensorflow.keras.callbacks import TensorBoard
import time
import numpy as np
import random


# permutations for loops
dense_layers = [0,1,2]
conv_layers = [2]
layer_sizes = [10,15,20]
batch_sizes = [10]
wl = [10]
data_blodck=[]
x = [1]

# folders by window length
for values in x:
    '''
    random.shuffle(data10RS);

    features = []
    labels = []
    for feature,label in data10RS:
        features.append(feature)
        labels.append(label)
```

```
    for wl in D:
        if  wl==10:
            Xa=  features[0:8300]
            Xta= features[8300:9880]
            ya=  labels[0:8300]
            tl=  labels[8300:9880]

    '''

    ftr = np.array(features)
    rftr = ftr.reshape(-1,72,10,1)



    for dense_layer in dense_layers:
        for layer_size in layer_sizes:
            for conv_layer in conv_layers:
                for bs in batch_sizes:
                    NAME =
f"thesis_quinary_stats_ep100_{wl}_wl_d1_{conv_layer}_conv_{layer_size}_nodes_{dense_la
yer}_dense_{bs}_batch_{int(time.time())}_time"
                    print(NAME)
                    tboard = TensorBoard(log_dir =
'binary_ten_eps_logs\{}'.format(NAME))

                    model = Sequential()

                    # first layer must have input shape
                    model.add(Conv2D(layer_size,(3,3),input_shape = rftr.shape[1:]))
                    model.add(Activation("relu"))
                    model.add(MaxPooling2D(pool_size=(2,2)))

                    for l in range(conv_layer-1):
                        model.add(Conv2D(layer_size,(3,3)))
                        model.add(Activation("relu"))
                        model.add(MaxPooling2D(pool_size=(2,2)))
                        model.add(Dropout(0.1))


                    # flattening before fully connected layers
                    model.add(Flatten())


                    for l in range(dense_layer):
                        model.add(Dense(layer_size))
                        model.add(Activation("relu"))

                    model.add(Dense(5))
                    model.add(Activation("sigmoid"))
                    model.compile(loss = "sparse_categorical_crossentropy",
                                optimizer = "adam",
                                metrics = ['accuracy'])

                    print("\nTRAINING : ")
                    print(NAME)
                    print("xoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxox")
                    model.fit(rftr,labels,batch_size=bs, validation_split=0.2,
epochs=100 , callbacks = [tboard])
                    print("xoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxoxox")
                    print(NAME)
                    print("COMPLETED\n\n")
                    model.save(NAME)
```

TESTING MODELS OVER IMPORTED TEST DATA-A AND B

A

```
import tensorflow as tf
import time
import numpy as np


#Xtab = np.array(features_test_2);
Xtab = np.array(features_test_1);
Xtab = Xtab.reshape(-1,72,10,1);
#modelname =
'QUINARY_stats_[10]_wl_d1_2_conv_15_nodes_1_dense_10_batch_1566013925_time';
model =tf.keras.models.load_model(modelname)
predicted_labels_1 = []

#predicted_labels_2 = []

data_block_1=[]
#data_block_2=[]
prediction_time_1 = []


for xxx in Xtab:
    xxy = xxx.reshape(-1,72,10,1)
    tima = time.time();
    ytab = model.predict(xxy)
    timb = time.time();
    pt = timb-tima
    prediction_time_1.append(pt)
    #prediction_time_2 = (timb- tima)

    for I in ytab:
        predicted_labels_1.append(np.argmax(i))
   #predicted_labels_2.append(np.argmax(i))


count=0;
test_accuracy=0;


o=range(0,len(predicted_labels_1))

for k in o:
    if predicted_labels_1[k] == labels_test_1[k]:
        count+=1
'''
o=range(0,len(predicted_labels_2))
for k in o:
    if predicted_labels_2[k] == labels_test_2[k]:
        count+=1
'''
test_accuracy=(count*100)/728
print(test_accuracy)


data_block_1.append([test_accuracy,modelname,prediction_time_1])
```

B

```
import tensorflow as tf
import time
import numpy as np
import statistics

#Xtab = np.array(features_test_2);
Xtab = np.array(features_test_2);
Xtab = Xtab.reshape(-1,72,10,1);
#modelname =
'QUINARY_stats_[10]_wl_d1_2_conv_15_nodes_1_dense_10_batch_1566013925_time';
model =tf.keras.models.load_model(modelname)
predicted_labels_1 = []

#predicted_labels_2 = []

data_block_2=[]
#data_block_2=[]
prediction_time_b = []


for xxx in Xtab:
    xxy = xxx.reshape(-1,72,10,1)
    tima = time.time();
    ytab = model.predict(xxy)
    timb = time.time();
    pt = timb-tima
    prediction_time_b.append(pt)
    #prediction_time_2 = (timb- tima)

    for i in ytab:
        predicted_labels_1.append(np.argmax(i))
   #predicted_labels_2.append(np.argmax(i))


count=0;
test_accuracy_b=0;


o=range(0,len(predicted_labels_1))

for k in o:
    if predicted_labels_1[k] == labels_test_2[k]:
        count+=1
'''
o=range(0,len(predicted_labels_2))
for k in o:
    if predicted_labels_2[k] == labels_test_2[k]:
        count+=1
'''
test_accuracy_b=(count*100)/60
print(test_accuracy_b)


data_block_2.append([test_accuracy_b,modelname,prediction_time_b])
#data_block_2.append([test_accuracy,ytab,prediction_time_2,labels_test_2,predicted_lab
els_2])


print(modelname)
```

```
average_accuracy = ((test_accuracy*728)+(test_accuracy_b*60))/(728+60);
average_time =
(((statistics.mean((prediction_time_1))*728)+((statistics.mean(prediction_time_b)*60))
)/(728+60));

print(average_accuracy);
print(average_time);
```

REFERENCES

[1] J. J. Daly and J. R. Wolpaw, "Brain-computer interfaces in neurological rehabilitation," *The Lancet Neurology*. 2008.

[2] J. R. Wolpaw and E. W. Wolpaw, *Brain-Computer Interfaces: Principles and Practice*. 2012.

[3] M. Cope and D. T. Delpy, "System for long-term measurement of cerebral blood and tissue oxygenation on newborn infants by near infra-red transillumination," *Med. Biol. Eng. Comput.*, 1988.

[4] A. Villringer and B. Chance, "Non-invasive optical spectroscopy and imaging of human brain function," *Trends Neurosci.*, 1997.

[5] L. F. Nicolas-Alonso and J. Gomez-Gil, "Brain computer interfaces, a review," *Sensors*. 2012.

[6] C. Vidaurre, M. Kawanabe, P. Von Bünau, B. Blankertz, and K. R. Müller, "Toward unsupervised adaptation of LDA for brain-computer interfaces," *IEEE Trans. Biomed. Eng.*, 2011.

[7] D. A. Boas, C. E. Elwell, M. Ferrari, and G. Taga, "Twenty years of functional near-infrared spectroscopy: Introduction for the special issue," *NeuroImage*, 2014.

[8] S. Tak and J. C. Ye, "Statistical analysis of fNIRS data: A comprehensive review," *NeuroImage*. 2014.

[9] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights Imaging*, vol. 9, no. 4, pp. 611–629, 2018.

[10] A. Torricelli *et al.*, "Assessment of cortical response during motor task in adults by a multimodality approach based on fNIRS-EEG, fMRI-EEG, and TMS," in *Optics InfoBase Conference Papers*, 2011.

[11] A. Torricelli *et al.*, "Cortical response during motor task in epileptic patients with movement disorders: A multimodality fNIRS-EEG, fMRI-EEG and TMS clinical study," in *Asia Communications and Photonics Conference, ACP*, 2012.

[12] D. T. Delpy and M. Cope, "Quantification in tissue near-infrared spectroscopy," *Philos. Trans. R. Soc. B Biol. Sci.*, 1997.

[13] N. Naseer and K.-S. Hong, “fNIRS-based brain-computer interfaces: a review,” *Front. Hum. Neurosci.*, 2015.

[14] S. Brigadoi *et al.*, “Motion artifacts in functional near-infrared spectroscopy: A comparison of motion correction techniques applied to real cognitive data,” *Neuroimage*, 2014.

[15] K.-S. Hong, M. J. Khan, and M. J. Hong, “Feature Extraction and Classification Methods for Hybrid fNIRS-EEG Brain-Computer Interfaces,” *Front. Hum. Neurosci.*, vol. 12, Jun. 2018.

[16] M. Ferrari and V. Quaresima, “A brief review on the history of human functional near-infrared spectroscopy (fNIRS) development and fields of application,” *NeuroImage*. 2012.

[17] S. Brigadoi *et al.*, “Motion artifacts in functional near-infrared spectroscopy: a comparison of motion correction techniques applied to real cognitive data,” *Neuroimage*, vol. 85, pp. 181–191, 2014.

[18] E. Kirlilna, N. Yu, A. Jelzow, H. Wabnitz, A. M. Jacobs, and L. Tachtsidis, “Identifying and quantifying main components of physiological noise in functional near infrared spectroscopy on the prefrontal cortex,” *Front. Hum. Neurosci.*, 2013.

[19] R. Venkatesan, B. Li, R. Venkatesan, and B. Li, “Convolutional Neural Networks,” in *Convolutional Neural Networks in Visual Computing*, 2018.

[20] S. W. Smith, “Properties of Convolution,” in *Digital Signal Processing*, 2003.

[21] T. Trakoolwilaiwan, B. Behboodi, J. Lee, K. Kim, and J.-W. Choi, “Convolutional neural network for high-accuracy functional near-infrared spectroscopy in a brain-computer interface: three-class classification of rest, right-, and left-hand motor execution,” *NEUROPHOTONICS*, vol. 5, no. 1, 2018.

[22] A. Lavin and S. Gray, “Fast Algorithms for Convolutional Neural Networks,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016.

[23] J. Shin *et al.*, “Open Access Dataset for EEG+NIRS Single-Trial Classification,” *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 25, pp. 1735–1745, 2017.

[24] J. Shin, A. Von Lühmann, D. W. Kim, J. Mehnert, H. J. Hwang, and K. R. Müller, “Data descriptor: Simultaneous acquisition of EEG and NIRS during cognitive tasks for an open access dataset,” *Sci. Data*, 2018.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems*, 2012.

[26] P. Y. Simard, D. Steinkraus, and J. C. Platt, “Best practices for convolutional neural networks applied to visual document analysis,” in *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*, 2003.

[27] J. Schmidhuber, “Deep Learning in neural networks: An overview,” *Neural Networks*. 2015.

FNIRS Data Classification for Brain Computer Interface Using Deep Learning

ORIGINALITY REPORT

9%	4%	6%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Jaeyoung Shin, Alexander von Luhmann, Benjamin Blankertz, Do-Won Kim, Jichai Jeong, Han-Jeong Hwang, Klaus-Robert Muller. "Open Access Dataset for EEG+NIRS Single-Trial Classification", IEEE Transactions on Neural Systems and Rehabilitation Engineering, 2016 Publication	3%

2	Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, Kaori Togashi. "Convolutional neural networks: an overview and application in radiology", Insights into Imaging, 2018 Publication	1%
3	Jaeyoung Shin, Alexander von Lühmann, Do Won Kim, Jan Mehnert, Han-Jeong Hwang, Klaus-Robert Müller. "Simultaneous acquisition of EEG and NIRS during cognitive tasks for an open access dataset", Scientific Data, 2018 Publication	1%
4	www.spiedigitallibrary.org Internet Source	1%
5	fedetd.mis.nsysu.edu.tw Internet Source	1%
6	deepbci.korea.ac.kr Internet Source	<1%
7	Noman Naseer, Keum-Shik Hong. "Decoding Answers to Four-Choice Questions Using Functional near Infrared Spectroscopy", Journal of Near Infrared Spectroscopy, 2015 Publication	<1%

8	Submitted to United World College of South East Asia Student Paper	<1%
9	en.wikipedia.org Internet Source	<1%
10	www.science.gov Internet Source	<1%
11	digital.lib.washington.edu Internet Source	<1%
12	Submitted to Loughborough University Student Paper	<1%
13	Submitted to University College London Student Paper	<1%
14	docplayer.net Internet Source	<1%

15	Submitted to The Hong Kong Polytechnic University Student Paper	<1%
16	Submitted to Eastern Mediterranean University Student Paper	<1%
17	Jaeyoung Shin, Do-Won Kim, Klaus-Robert Müller, Han-Jeong Hwang. "Improvement of Information Transfer Rates Using a Hybrid EEG-NIRS Brain-Computer Interface with a Short Trial Length: Offline and Pseudo-Online Analyses", Sensors, 2018 Publication	<1%
18	Submitted to University of Wollongong Student Paper	<1%
19	Submitted to University of Gloucestershire Student Paper	<1%

20	hal.umontpellier.fr Internet Source	<1%
21	Submitted to Minnetonka High School Student Paper	<1%
22	Submitted to Sabanci Universitesi Student Paper	<1%
23	Yu Zhang, Xiong Zhang, Han Sun, Xuefei Zhong, Zhaowen Fan. "A Wearable Wireless fNIRS System", Proceedings of the 2018 8th International Conference on Bioscience, Biochemistry and Bioinformatics - ICBBB 2018, 2018 Publication	<1%
24	"Advances in Neuroergonomics and Cognitive Engineering", Springer Science and Business Media LLC, 2020 Publication	<1%

Digital Receipt

turnitin

Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author:	Ahmad Subhani
Assignment title:	Plagiarism Detection 2019 Part 1 (M
Submission title:	FNIRS Data Classification for Brain
File name:	88741Ahmad Subhani FNIRS Da
File size:	3.32M
Page count:	36
Word count:	6,650
Character count:	35,608
Submission date:	05-Nov-2019 01:49PM (UTC+0500)
Submission ID:	1207442106

Abstract

Brain Computer Interfaces (BCIs) translate recorded brain data directly to machine commands that can be used to control external devices. They are composed of three different functions i.e. recording of data from the brain, processing of data to recognize the intention of the subject and translation of the data into appropriate command for the machine being controlled. Functional near-infrared spectroscopy (fNIRS) is among one of the brain signal recording techniques which uses near-infrared spectroscopy (NIRS) for functional neuroimaging. It uses near-infrared light wavelengths (between 650 and 1000 nm) to measure the optical absorption changes of brain tissues. Use of fNIRS for BCIs limited because of slow hemodynamic response to stimulus, blood flow in scalp and undeveloped techniques for classifying signals.

In this thesis we train Convolutional Neural Networks to classify fNIRS signals for BCIs. These networks classify raw signals with more than 95% testing accuracy for cognitive and imagery tasks with upto five separate categories in less than two milliseconds, (dependent on the processing power available), thus showing promising improvement in current classification efficiency.