Development of An Application for Combining Disparate EEG Seizure Datasets into A Single Dataset

By

IKEZOGWO O. WISDOM

EEG/2013/051

A THESIS SUBMITTED TO
DEPARTMENT OF ELECTRONIC & ELECTRICAL ENGINEERING
FACULTY OF TECHNOLOGY
OBAFEMI AWOLOWO UNIVERSITY, ILE-IFE.

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR
THE AWARD OF BACHELOR OF SCIENCE IN
ELECTRONIC & ELECTRICAL ENGINEERING

December 4, 2019

Department of Electronic and Electrical Engineering,

Obafemi Awolowo University,

Ile-Ife, Osun State.

December 4, 2019

The Final Year Project Coordinator,

Department of Electronic and Electrical Engineering,

Obafemi Awolowo University,

Ile-Ife, Osun State.

Dear Sir,

LETTER OF TRANSMITTAL

In partial fulfillment for the award of a B.Sc in Electronic and Electrical Engineering and in response to your request for the submission of a report on my final year project, I hereby submit this report to you. The report presents a detailed description of the project as well as results recorded during the course of implementation.

Yours faithfully,

IKEZOGWO Wisdom Oluchineke

EEG/2013/051

Certification

I, Wisdom Oluchineke IKEZOGWO with registration number EEG/2013/051 in the Department of Electronic and Electrical Engineering, Faculty of Technology, Obafemi Awolowo University certify that this is an original research carried out under the supervision of:

Dr. Kayode P. Ayodele

Dedication

I dedicate this work to God Almighty and my family who have always been there for me. To my Father and Mother Mr and Mrs Ikezogwo, To my Supervisor Dr. Ayodele I say thank you.

I also would like to thank all that have supported me over the years from my colleagues to my lectures with the following be the people of note: Dr. Ilori, Prof. L.O Kehinde, Dr Akinwale, Dr Jubril, Dr. Ogunba, Dr. Osuntuyi, Dr Ariyo, Mr Pipe, Mr Jide Omotola, Mr Lawal AbdulKabir, Ms Boluwatife, Ms Odufuwa Moyinoluwa, Mr Babatunde Samuel.

Acknowledgment

The undergraduate experience I had up till now at Obafemi Awolowo University has been great chance for learning and professional development. Therefore, I consider myself as a very lucky individual as I was provided with an opportunity to be a part of it. I am also grateful for having a chance to meet so many wonderful people and professionals who led me through this period.

Bearing in mind previous I am using this opportunity to express my deepest gratitude and special thanks to the Head of Departments, present and past, and all the entire staff of the department, both teaching and non-teaching. Also, I express my deepest thanks to all my colleagues who have contributed one way or the other to my success in the department. I also express my sincere appreciation to my parents who are always there for me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Epilepsy is a neurological disorder affecting more than 50 million people worldwide. Monitoring of brain activity through electroencephalograms (EEG) is the standard technique for the diagnosis of epilepsy. In current clinical practice, EEG readings must be analyzed by trained neurologists to identify characteristic patterns of the disease, such as seizures and pre-ictal spikes. However this visual analysis is extremely laborious, taking several hours to analyze one day of recording from a single patient, and it requires scarce highly trained professionals.

Neural networks refers to artificial neural networks, which consists of artificial neurons or nodes. Thus the term may refer to either biological neural networks, made up of real biological neurons, or artificial neural networks, for solving artificial intelligence (AI) problems.The connections of the biological neuron are modeled as weights. A positive weight reflects an excitatory connection, while negative values mean inhibitory connections. All inputs are modified by a weight and summed. This activity is referred as a linear combination. Finally, an activation function controls the amplitude of the output. For example, an acceptable range of output is usually between 0 and 1, or it could be -1 and 1. Unlike the von Neumann model computations, artificial neural networks do not separate memory and processing and operate via the flow of signals through the net connections, somewhat akin to biological networks. These artificial networks may be used for predictive modeling, adaptive control and applications where they can be trained via a dataset. There have been dramatic improvements in the performance of neural networks over the last two decades. In particular, multiple-layer-deep networks have demonstrated impressive performance in tasks in the domains of audio, text, image, and video data. Generally, when sufficiently large datasets are available, and combined with sufficiently-sophisticated deep networks, performances have exceeded those of humans.

Despite numerous successful applications of deep neural networks to large-scale image, video and text data, they remain relatively unexplored in neuroimaging domain. Perhaps one of the main reasons here is that the number of samples in most neuroimaging datasets is limited, thus making such data less adequate for training large-scale networks with millions of parameters.

There are a number of possible approaches to solving this problem. Data augmentation techniques such as generative adversarial networks have been used with mixed results. Secondly, there are attempts to place much larger quantities of EEG recordings online. This project proposes another possibly unusual approach: combining disparate datasets. Generally speaking, data from different EEG datasets are acquired under very different settings, and with differing parameters. Combining data from such different populations for the purpose of training neural networks will thus have the obvious pitfall of significant label and feature noise.

1.2 Problem Statement

Over the past few years, there have been tremendous improvements in the performances of neural networks , this is predominantly due to the massive increase in the size of data (Structured or Unstructured) available. In particular, Multiple-layer Deep Neural Network (DNN) models have demonstrated impressive performance in several task in domains that include but are not limited to Audio, Image, Text and Video data.

These successes can be attributed to the massive amount of data points used in training these models, such large data corpus which include: ImageNet as described in (Krizhevsky, Sutskever, and Geoffrey E., 2012), Youtube-8M video dataset as described in (Chen, 2017), AUDIO-SET an audio dataset as described in (Gemmeke et al., 2017) etc., drive the performance of these models beyond those set by humans. One research domain that stands to benefit greatly from the creation of such data corpus is Neuroimaging. Unfortunately, the lack of sufficient data population in this domain restricts such models from exceeding human performance when applied for inference.

Therefore the need to investigate the viability of transforming appropriate parameters from different EEG epileptic seizure datasets such that the feature noise resulting from their combination is reduced.

1.3 Research Motivation

The motivation for this problem is both broad and specific. Specialized image and video classification tasks often have insufficient data. This is particularly true in the medical industry, where access to data is heavily protected due to privacy concerns. Ideally publicly-available EEG data should be sufficient to drive several: studies to be made, hypothesis to be tested and models to generalize their function to new data-points This ideal dataset should be be created such that new data-points can be appended and outliers removed.

There are several disparate available for use, however none of these have proven to be individually sufficient to fuel state-of-the-art research. In other to fill the need identified, an application or technique needs to be developed for combining these datasets

1.4 Aims and Objectives

The aim of this project is to develop a technique and application that will process EEG seizure data from different datasets (sources, sessions, or studies) so that they can be treated as being part of a single dataset. The above aim would be achieved by fulfilling the following research objectives:

1. Identify and catalog every publicly-available EEG epileptic seizure database around the world.

2. Determine every parameter of importance that can vary from dataset to dataset.
3. Determine, with appropriate justification, which subset of the parameters in (2) are likely to impact the performance of seizure detector algorithms
4. Identify which of the parameters in (2) can be modified by third parties not privy to the data acquisition sessions.
5. Determine reference values, frameworks or scales for each of the parameters in (3).
6. Develop a Normalizer computer application in the Python programming language to modify EEG data from each EEG dataset.
7. Demonstrate the programme developed by obtaining multiple publicly-available datasets and processing this datasets.
8. Identify and obtain existing epileptic seizure detection model.
9. Using the identified model, determine the classification accuracy of each dataset.
10. Using the identified model, determine the classification accuracy of the combined dataset.

1.5 Scope of The Project

This work has been optimized for electroencephalography (time-series data) and not any other variant of neuroimaging data. In this study the the method proposed by (Bashivan et al., 2015) is applied.

A domain generalization approach is explored. By combining disparate datasets to increase the quantity of training data, dataset bias can be reduced simultaneously. The challenge is to achieve this while minimizing covariate shifts that would otherwise impact performance, which is the focus of this thesis.

Chapter 2

Literature Review

2.1 Electroencephalography

Electroencephalogram (EEG) is an effective and non-invasive technique commonly used for monitoring the brain activity and diagnosis of epilepsy (Ullah et al., 2010). Due to capability to reflect both the normal and abnormal electrical activity of the brain, EEG has been found to be a very powerful tool in the field of neurology and clinical neurophysiology.

2.1.1 10-20 System

The International Federation of Societies for Electroencephalography and Clinical Neurophysiology has recommended the conventional electrode setting (also called 10–20) for 21 electrodes (excluding the earlobe electrodes), as depicted in Figure 2.1. Often the earlobe electrodes called A1 and A2, connected respectively to the left and right earlobes, are used as the reference electrodes. The 10–20 system avoids both eyeball placement and considers some constant distances by using specific anatomic landmarks from which the measurement would be made and then uses 10 or 20% of that specified distance as the electrode interval. The odd electrodes are on the left and the even ones on the right. For setting a larger number of electrodes using the above conventional system, the rest of the electrodes are placed in between the above electrodes with equidistance between them.

Two different modes of recordings, namely differential and referential, are used. In the differential mode the two inputs to each differential amplifier are from two electrodes. In the referential mode, on the other hand, one or two reference electrodes are used. Several different reference electrode placements can be found in the literature. Physical references can be used as vertex (Cz), linked-ears, linked-mastoids, ipsilateral ear, contralateral ear, C7, bipolar references, and tip of the nose [28]. There are also reference-free recording techniques, which actually use a common average reference. The choice of reference may produce topographic distortion if the reference is not relatively neutral. In modern instrumentation, however, the choice of a reference does not play an important role in the measurement. In such systems other references such as FPz, hand, or leg electrodes may be used. The overall setting includes the active electrodes and the references. In another similar setting, called the Maudsley electrode positioning system, the conventional 10–20 system has been modified to capture better the signals from epileptic foci in epileptic seizure recordings. The only difference between this system and the 10–20 conventional system is that the outer electrodes are slightly lowered to enable better capturing of the required signals. The advantage of this system over the conventional one is that it provides a more extensive coverage of the lower part of the cerebral convexity, increasing the sensitivity for the recording from basal subtemporal structures, as adapted from (Sanei and Chambers, 2007)

2.1.2 Parameters and Settings

The parameters of the EEG time-series are can be seen as the pre-acquisition and post-acquisition chacracteristics of the EEG data. These are the variables that determine how different each dataset is to each other, it also serves as the systems fuction which can be transformed. see Table 2.1

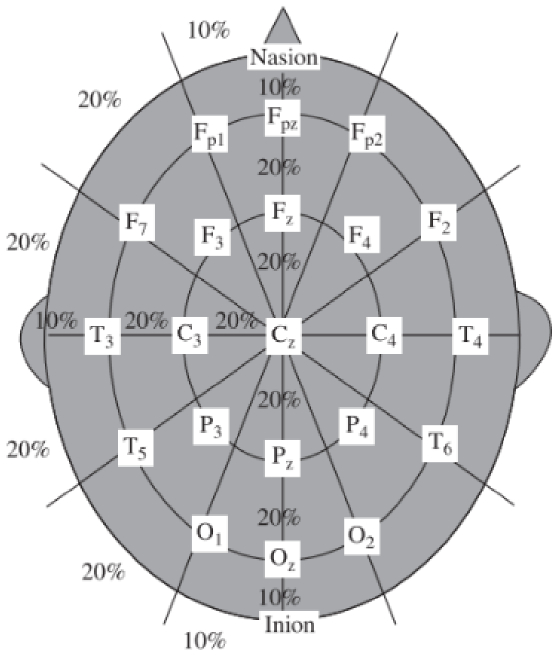

Figure 2.1: Conventional 10–20 EEG electrode positions for the placement of 21 electrodes (Sanei and Chambers, 2007)

Table 2.1: EEG Parameters and Characteristic Settings.

Param	**Bonn**	**CHB-MIT**	**Bern**	**Epilepsiae**	**AESS**	**TUH**
Sampling rate	173.61 Hz	256 Hz	512 Hz	250 Hz - 2.5KHz	400 Hz(man), 5KHz(dog)	256 Hz
No. of Trials		23				
Samples/Trial			10240			
Duration of Trial	23.6s		20s			
Total Time	35 hrs	977 hrs		40,000 hrs		
Amplitude range						
No. of Electrodes						
Location	10-20 system	10-20 system	10-20 system	10-20 system	10-20 system	10-20 system
Montage						TCP Montage
Preprocessing	BPF 0.53-40 Hz	BPF 2-20 Hz	BPF 0.5-150 Hz			
No. of Channels	100 Single	22	64	126	16	19
Resolution	12 Bits	16 Bits				
No. of Patients		22	5	275	7	246
No. of Seizures				58-189	48	
Inter-ictal Interval		209 hrs		¿4 hrs	627.7 h	
Avg. Seizure/Patient				3		

FIR	4th Order	4th Order				
EEG Type		Scalp	Intra		Intra**	
Size						
Format		EDF				
Focal Oriented	Yes		Yes			
Adults						

2.1.3 EEG Seizure Dataset Description

1. **University of Boon Dataset**:

   This dataset was acquired by (Andrzejak et al., 2001) the Epilepsy Centre at the University of Bonn, Germany. All EEG signals were recorded with the same 128-channel amplifier system, using an average common reference omitting electrodes containing pathological activity C, D, and E or strong eye movement artifacts A and B. After 12 bit analog-to-digital conversion, the data were written continuously onto the disk of a data acquisition computer system at a sampling rate of 173.61 Hz. Band-pass filter settings were 0.53– 40 Hz 12 dB/Oct. It contains 35 hours of time series data.

   The EEG signals were recorded using standard 10-20 electrode placement system. The complete data consists of five sets (A to E), each containing 100 one-channel instances. Sets A and B consists of EEG signals recorded from five healthy volunteers while they were in a relaxed and awake state with eyes opened (A) and eyes closed (B), respectively.Sets C, D, and E were recorded from five patients. EEG signals in set D were taken from the epileptogenic zone. Set C was recorded from the hippocampal formation of opposite hemisphere of the brain. Sets C and D consist of EEG signals measured during seizure-free intervals (interictal), whereas, the EEG signals in Set E were recorded only during seizure activity (ictal) (Andrzejak et al., 2001).

2. **CHB-MIT Dataset**:

   EEG time series data made available online by the Children's Hospital Boston, consisting of EEG recording of subjects with intractable seizures are also considered. Data sets available are filtered already using hardware filters 0.5-70 Hz and 50 Hz notch filters. Recordings are from 22 subjects (5 males, ages 3-22 and 17 females, ages 1.5-18). Mean (Standard Deviation) ages of the patients are 9.81 (5.75 years). The International 10-20 system of EEG electrode positions and nomenclature are used for 23 channels. These recordings are FP1-F7, F7-T7, T7-P7, P7-O1, FP1-F3, F3-C3, C3-P3, P3-O1, FP2-F4, F4-C4, C4-P4, P4-O2, FP2-F8, F8-T8, T8-P8, P8-O2, FZ-CZ, CZ-PZ, P7-T7, T7-FT9, FT9-FT10, FT10-T8 and T8-P8. All signals are sampled at 256 samples per second with 16-bit resolution and EEG data recorded is exactly one hour of digitized signal duration, some are of two hour duration and some other 4 h duration. The conditions namely non seizure and seizure and the duration are specified in the database itself. This database consists of EEG recordings from pediatric subjects with intractable seizures. Subjects were monitored for up to several days following withdrawal ofanti-seizure medication in order to characterize their seizures and assess their candidacy for surgical intervention. Recordings, grouped into 23 cases, were collected from 22 subjects (5 males, ages 3–22; and 17 females, ages 1.5–19). (Case chb21 was obtained 1.5 years after case chb01, from the same female subject).

In all, these records include 198 seizures (182 in the original set of 23 cases); the beginning ([) and end (]) of each seizure is annotated in the .seizure annotation files that accompany each of the files listed in RECORDS-WITH-SEIZURES. In addition, the files named chb*nn*-summary.txt contain information about the montage used for each recording, and the elapsed time in seconds from the beginning of each .edf file to the beginning and end of each seizure contained in it.

3. **Bern-Barcelona EEG Dataset**:

The Bern Barcelona EEG database is a public domain database of intracranial electroencephalography recordings from patients with epilepsy. It was published by the researchers R.G. Andrzejak, C. Rummel, and K. Schindler in October 2012 along with their scientific article Nonrandomness, nonlinear dependence, and nonstationarity of electroencephalographic recordings from epilepsy patients. In this article the authors (Andrzejak, Schindler, and Rummel, 2012) applied nonlinear signal analysis techniques to these FFG recordings. Detailed results as well as computer source codes used to derive these results are also included in the database. The datasetis hosted at the pages of the non-linear signal analysis group at the University at Pompeu Fabra. The human data are from patients with temporal and extratemporal lobe epilepsy undergoing evaluation for epilepsy surgery. The iEEG recordings are from depth electrodes implanted along anterior-posterior axis of hippocampus, and from subdural electrode grids in various locations. Data sampling rates vary from 500 Hz to 5,000 Hz. The canine data are from an implanted device acquiring data from 16 subdural electrodes. Two 4-contact strips are implanted over each hemisphere in an antero-posterior orientation. Data are recorded continuously at a sampling frequency of 400 Hz and referenced to the group average. Includes intracranial EEG recordings from five epilepsy patients. These recordings were performed prior to and independently from our study as part of the epilepsy diagnostics in these patients. All five patients had long standing pharmacoresistant temporal lobe epilepsy and were candidates for epilepsy surgery. Noninvasive studies had not allowed for unequivocal localization of the brain areas from which seizures originated ("seizure onset zone"), and all patients underwent long-term intracranial EEG recordings at the Department of Neurology of the University of Bern. Multichannel EEG signals were recorded with intracranial strip and depth electrodes all manufactured by AD-TECH (Racine, WI, USA). An extracranial reference electrode placed between 10/20 positions Fz and Pz was used. EEG signals were either sampled at 512 or 1024 Hz, depending on whether they were recorded with more or less than 64 channels. All EEG signals were digitally band-pass filtered between 0.5 and 150 Hz using a fourth-order Butterworth filter. Forward and backward filtering was used in order to minimize phase distortions. Those EEG signals that had been recorded with a sampling rate of 1024 Hz were down-sampled to 512 Hz prior to further analysis.

4. **Temple University Hospital Seizure Dataset (TUH)**: To build an annotated seizure dataset, we first needed an abundant source of EEG data. Our work here utilized a subset which includes approximately 90% of v0.6.0 of TUH EEG. The data is organized by patient and by session. Each session contains EEG signal data stored in a standard European Data Format (EDF).

   The most recent release of TUSZ is v1.2.0, which was released in December 2017. It contains 315 subjects with a total of 822 sessions, of which 280 sessions contain seizures. Each file is completely transcribed in two ways: channel-based and term-based. A channel-based annotation refers to labeling of the start and end time of an event on a specific channel. A term-based annotation refers to a summarization of the channel-based annotations – all channels share the same annotation, which is an aggregation of the per-channel annotations. The entire seizure database has been divided into training and evaluation sets to support machine learning research. All files in this corpus are pruned versions of the original EEG recordings. The duration of a single pruned file is no more than one hour. The training and evaluation sets contain 265 and 50 subjects respectively. The patients in the evaluation set were selected based on gender (56% of the patients in the evaluation set are female; 50% female in the training set) and selected to maximize a number of demographic features.

5. **EPILEPSIAE Dataset**:

   The EPILEPSIAE database is by far the largest and most comprehensive database for human surface and intracranial EEG data.

   The database consist of surface recordings from 217 patients and also invasive EEG recordings from 58 patients. The number of seizures per patient range from 3 to 94. Surface recordings were performed by a 10–20 electrode scheme, although for some patients it was extended by additional electrode contacts. For patients with invasive recordings stereotactically implanted depth electrodes were used, including up to 125 electrode channels for some patients. Sampling rates range from 250Hz to 2.5kHz. In the case of intra-cranial recordings, three-dimensional coordinates of the electrodes according to the Montreal Neurologic Institute (MNI) coordinate system (Evans et al., 1993) are provided for each patient. It consists of datasets of 275 patients and comprises 2,662 seizures.

6. **UPenn and Mayo Clinic's Seizure Dataset (AESS)**:

   Intracranial EEG data analyzed for this study was provided by the UPenn-Mayo Clinic Seizure Detection Challenge on kaggle.com, sponsored by the American Epilepsy Society. The data consisted in 1s long multi-channel intracranial EEG recordings (segments) from 4 dogs and 8 human patients. Each patient had a fixed number of implanted electrodes from which signal was recorded (N channel) at a fixed sampling rate fs. No information was given about the localization of the electrodes. Data are organized in folders containing training and testing data for each human or canine subject. The training data is organized into 1-second EEG clips labeled ”Ictal” for seizure data segments, or ”Interictal” for non-seizure data segments. Training data are arranged sequentially while testing data are in random order. Ictal training and testing data segments are provided covering the entire seizure, while interictal data segments are provided covering approximately the mean seizure duration for each subject. Starting points for the interictal data segments were chosen randomly from the full data record, with the restriction that no interictal segment be less than one hour before or after a seizure. The human data are from patients with temporal and extratemporal lobe epilepsy undergoing evaluation for epilepsy surgery. The iEEG recordings are from depth electrodes implanted along anterior-posterior axis of hippocampus, and from subdural electrode grids in various locations. Data sampling rates vary from 500 Hz to 5,000 Hz. The canine data are from an implanted device acquiring data from 16 subdural electrodes. Two 4-contact strips are implanted over each hemisphere in an anteroposterior orientation. Data are recorded continuously at a sampling frequency of 400 Hz and referenced to the group average.

Table 2.2: Papers describing the dataset

Dataset	**Paper**	**Subject**
CHB-MIT	(Shoeb and Guttag, 2010)	Provides descriptive information about the CHB-MIT dataset whilst using the data for a study into the *Application of Machine Learning To Epileptic Seizure Detection*
	(Goldberger et al., 2000)	The purpose of this article is to provide a brief introduction to the newly established Research Resource for Complex Physiologic Signals and to invite participation by the bio- medical community in a cooperative research enterprise
	Detti paper-here	This paper on *Anticipating epileptic seizures through the analysis of EEG synchronization as a data classification problem* provides a near-perfect description of the dataset.
	(Theodorakopoulou, 2017)	A study on *Machine learning data preparation for epileptic seizures prediction* gives a primer of the EEG seizure dataset.
	(Shoeb and Smith, 2004)	This Msc thesis was one of the earliest study to cite and use the dataset as a resource for work on *Patient-Specific Seizure Onset Detection.*

Bern	(Andrzejak, Schindler, and Rummel, 2012)	Initial study at the University of Bern describing the data and also the study of *Nonrandomness, nonlinear dependence, and nonstationarity of electroencephalographic recordings from epilepsy patients*
	(Chen, Wan, and Bao, 2016)	In this work, the epileptic focus localization is formulated into a classification problem: classifying “focal” and “non-focal” EEG channels from multi-channel EEG recordings. The iEEG dataset was used test their DWT solution to the problem of epileptic focus localization.
	(Sriraam and Raghu, 2017)	Provides vital description of the dataset in the process of *Classifying Focal and Non Focal Epileptic Seizures Using Multi-Features and SVM Classifier*
	(Itakura, Tanaka, and Dataset, 2017)	Gives insight into the primary characteristics of the dataset as they try to solve the problem of *Epileptic Focus Localization Based on Bivariate Empirical Mode Decomposition and Entropy.*
	(Gupta et al., 2017)	They explore the dataset for a study on *Automated Detection of Focal EEG Signals using Features Extracted from Flexible Analytic Wavelet Transform*

TUH	(Shah et al., 2018)	This recent paper accomapnies the dataset as a defacto source for descriptive information on *The Temple University Hospital Seizure Detection Corpus.*
	(Shah et al., 2017)	In this study, we investigate the performance of a deep learning algorithm, CNN-LSTM, on several channel configurations. Providing more information about the target dataset.
	(Ziyabari, Shah, and Golmohammadi, 2017)	In this paper, they discuss the deficiencies of existing metrics for a seizure detection task and propose several new metrics that offer a more balanced view of performance. They demonstrate these metrics on a seizure detection task based on the TUH EEG Corpus.
	(Weltin et al., 2018)	In annotation of seizure events in the TUH EEG Seizure Corpus, independent slowing events were identified as a major source of false alarm error.The TUH EEG Slowing database, a subset of the TUH EEG Corpus, was created, and is introduced here, to aid in the development of a seizure detection
	(Roy, Kiral-kornek, and Harrer, 2018)	They focus on the Initial step in the process of EEG interpretation: Identification of abnormal and normal activity. Using a RNN the explore this new target dataset.

Bonn	(Andrzejak et al., 2001)	Analyzed sets of electroencephalographic EEG time series: surface EEG recordings from healthy volunteers with eyes closed and eyes open, and intracranial EEG recordings from epilepsy patients during the seizure free interval from within and from outside the seizure generating area as well as intracranial EEG recordings of epileptic seizures.
	(Ullah et al., 2010)	This study proposes a system based on deep learning, which is an ensemble of pyramidal one-dimensional convolutional neural network (P-1D-CNN) models. The models developed was then tested on the target Boon dataset and descriptive information was provided.
	(Chen, Wan, and Bao, 2016)	In this work, the epileptic focus localization is formulated into a classification problem: classifying “focal” and “non-focal” EEG channels from multi-channel EEG recordings. The target iEEG dataset was explored and descriptive information provided by the authors.

Epilepsia	(Truong et al., 2018)	This paper on *Convolutional Neural Networks for Seizure Prediction Using Intracranial and Scalp Electroencephalogram* provides insight into the target sub-dataset of the Epilepsiae Corpus, The Freiburg Hospital EEG seizure dataset.
AESS (Kaggle)	(Samie, Paul, and Bauer, 2018)	In this paper, the authors present an efficient and accurate algorithm for epileptic seizure prediction on low-power and portable IoT devices, leveraging on the target dataset for inference. *Highly Efficient and Accurate Seizure Prediction on Constrained IoT Devices.*
	(Truong et al., 2018)	This paper on *Convolutional Neural Networks for Seizure Prediction Using Intracranial and Scalp Electroencephalogram* provides insight into the target AESS seizure dataset.

2.2 Deep learning

Deep learning is a subfield of machine learning that has evolved out of the traditional approaches to artificial neural networks. Artificial neural networks are computational systems originally inspired by the human brain. They consist of many computational units, called neurons, which perform a basic operation and pass the information of that operation to further neurons. The operation is generally a summation of the information received by the neuron followed by the application of a simple, non-linear function. In most neural networks, these neurons are then organized into units called layers. The processing of neurons in one layer usually feeds into the calculations of the next, though certain types of networks will allow for information to pass within layers or even to previous layers. The final layer of a neural network outputs a result, which is interpreted for classification or regression. Figure 1 shows a depiction of the structure of a simple neural network. Deep Learning is a set of techniques that are a natural progression of traditional neural network techniques. These include: Stochastic Gradient Descent (SGD) algorithm and its descendants (e.g., Bottou, 2010). Gradient descent is a first-order iterative optimization algorithm used for finding the minimum of a function. In neural networks, it is used in conjunction with backpropagation to update the weights in the network.

2.2.1 Architectures

Perhaps the biggest difference between traditional neural networks and deep learning is the adoption of new types of layers in the network. Traditional neural network research focused on fully connected layers, in which every neuron in one layer is connected to every neuron in the next. While many of these layer types existed in the past, they usually were not able to used to significant effect due to various issues in training.

Table 2.3: Neural Network Architectures in Deep learning

Architecture	Description
CNN	CNNs are inspired by the organization of cat's visual cortex [85]. CNNs rely on local connections and tied weights across the units followed by feature pooling (subsampling) to obtain translation invariant descriptors [52]. The basic CNN architecture consists of one convolutional and pooling layer, optionally followed by a fully connected layer for supervised prediction. In practice, CNNs are composed by > 10 convolutional and pooling layers to better model the input space. The most successful applications of CNNs were obtained in computer vision [43, 44]. CNNs usually require a large data set of labeled documents to be properly trained.
RNN	RNNs are useful to process streams of data [53]. They are composed by one network performing the same task for every element of a sequence, with each output value dependent on the previous computations. In the original formulation, RNNs were limited to look back only a few steps owing to vanishing and exploding gradient problems. LSTM [86] and GRU [87] networks addressed this problem by modeling the hidden state with cells that decide what to keep in (and what to erase from) memory given the previous state, the current memory and the input value. These variants are efficient at capturing long-term dependencies and led to excellent results in Natural Language Processing applications [46, 47].

RBM	A RBM is a generative stochastic model that learns a probability distribution over the input space [54]. RBMs are a variant of Boltzmann machines, with the restriction that their neurons must form a bipartite graph. Pairs of nodes from each of the two groups (i.e. visible and hidden units) can have a symmetric connection between them, but there are no connections between nodes within a group. This restriction allows for more efficient training algorithms than the general class of Boltzmann machines, which allows connections between hidden units. RBMs had success in dimensionality reduction [55] and collaborative filtering [88]. Deep learning systems obtained by stacking RBMs are called Deep Belief Networks [89].
AE	An AE is an unsupervised learning model where the target value is equal to the input [55]. AEs are composed by a decoder, which transforms the input to a latent representation, and by a decoder, which reconstructs the input from this representation. AEs are trained to minimize the reconstruction error. By constraining the dimension of the latent representation to be different from input (and consequently from the output), it is possible to discover relevant patterns in the data. AEs are mostly used for representation learning [21] and are often regularized by adding noise to the original data (i.e. denoising AEs [90]).

2.3 Data Augmentation

2.3.1 Traditional Transformations

Traditional transformations consist of using a combination of affine transformations to manipulate the training data (Wang and Perez, 2017). For each input image, we generate a "duplicate" image that is shifted, zoomed in/out, rotated, flipped, distorted, or shaded with a hue. Both image and duplicate are fed into the neural net. For a dataset of size N, we generate a dataset of $2N$ size. see Figure 2.2

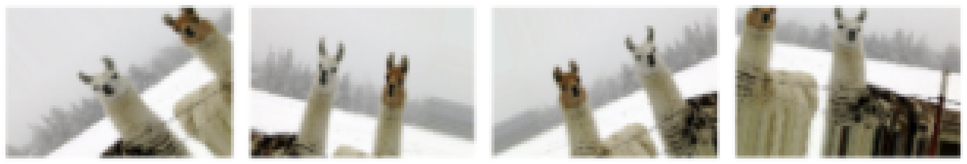

Figure 2.2: Traditional Transformation as adapted from (Wang and Perez, 2017)

2.3.2 Generative Adversarial Networks

The GAN framework consists of two opposing networks trying to outplay each other (Goodfellow et al., 2014). The first network, the discriminator, is trained to distinguish between real and fake input data. The second network, the generator, takes a latent noise variable z as input and tries to generate fake samples that are not recognized as fake by the discriminator. This results in a minimax game in which the generator is forced by the discriminator to produce ever better samples. One big drawback of GANs is the notorious instability of the discriminator during training. The discriminator might collapse into only recognizing few and narrow modes of the input distribution as real, which drives the generator to produce only a limited amount of different outputs. Mode collapse is very problematic for training GANs and is the subject of various works (Mao et al., 2016; Arjovsky et al., 2017; Gulrajani et al., 2017; Kodali et al., 2017).

While large parts of machine learning deal with the decoding of information from real-world data such as in classification tasks, there is also the recently very active field of how to generate such real-world data through implicit generative models. Generating artificial data could for example be used for data augmentation by producing natural looking samples that are not included in the original data set and thereby artificially increase training data with unseen samples. Ad- ditionally, the possibility to produce natural looking samples with certain properties, and the investigation of the models creating them, can be a useful tool for understanding the original data distribution used for training the GAN One recently proposed framework for the generation of artificial data are generative adversarial networks (Good- fellow et al., 2014) which showed groundbreaking results for the generation of artificial images.

GANs also allow the intentional manipulation of specific properties in generated samples (Radford et al., 2015) and therefore could prove to be a useful tool in understanding the original data distribution used for training the GAN. GANs have mainly been developed and applied to the generation of images and only a few studies investigating time series were conducted; recently they showed promising results for the generation of artificial audio (Donahue et al., 2018). The generation of artificial EEG signals would have applications in many different areas dealing with decoding and understanding brain signals, one recent study showed promising results in generating artificial EEG data (Hartmann, Schirrmeister, and Ball, 2018).

2.4 Data Integration

Data integration involves combining data residing in different sources and providing users with a unified view of them. This process becomes significant in a variety of situations, which include both commercial (such as when two similar companies need to merge their databases) and scientific (combining research results from different bioinformatics repositories, for example) domains. Data integration appears with increasing frequency as the volume (that is, big data) and the need to share existing data explodes. It has become the focus of extensive theoretical work, and numerous open problems remain unsolved.

2.4.1 The Problem of Integration

Integration of multiple information systems generally aims at combining selected systems so that they form a unified new whole and give users the illusion of interacting with one single information system. Users are provided with a homogeneous logical view of data that is physically distributed over heterogeneous data sources. For this, all data has to be represented using the same abstraction principles (unified global data model and unified semantics). This task includes detection and resolution of schema and data conflicts regarding structure and semantics.

In general, information systems are not designed for integration. Thus, whenever integrated access to different source systems is desired, the sources and their data that do not fit together have to be coalesced by additional adaptation and reconciliation functionality. Note that there is not the one single integration problem. While the goal is always to provide a homogeneous, unified view on data from different sources, the particular integration task may depend on:

- The kind of information that is managed by component systems (alphanumeric data, multimedia data; structured, semi-structured, unstructured data).

- requirements concerning autonomy of component systems, intended use of the integrated information system (read-only or write access),
- Performance requirements

Semantic Integration

Integration is more than just a structural or technical problem. Technically, it is rather easy to connect different relational DBMS. More demanding is to integrate data described by different data models; even worse are the problems caused by data with heterogeneous semantics. For instance, having only the name "loss" to denote a relation in an enterprise information system does not provide sufficient information to doubtlessly decide whether the represented loss is a book loss, a realized loss, or a future expected loss and whether the values of the tuples reflect only a roughly estimated loss or a precisely quantified loss. Integrating two "loss" relations with (implicit) heterogeneous semantics leads to erroneous results and completely senseless conclusions. Therefore, explicit and precise semantics of integratable data are essential for semantically correct and meaningful integration results.

2.4.2 Data Fusion

Data fusion is the analysis of several datasets such that different datasets can interact and inform each other.

2.5 Core Related Work

Data Augmentation techniques have been applied to visual data to the training data by artificial new images using techniques like moving windows, scaling, affine distortions, and elastic deformations and have resulted in increased performance of models leveraging the technique. In the past it had been difficult to apply such transformations to EEG data until a recent study by (Krell and Kim, 2014) which proposes a rotational data augmentation technique. Their results shows increased performance of signal processing chains for EEG-based brain-computer interfaces when rotating only around y- and z-axis with an angle around ±18 degrees to generate new data, This is however still very limiting in the total size of new data that can be created.

The technique proposed later in this paper is inspired by a recent study by (Bashivan et al., 2015) and (Thodoroff, Pineau, and Lim, 2016) convert EEG (time-series) data into images. This transformation i key to our proposed technique as it simplifies the spatial filtering process and reduces the complexity of the problem.

This project also borrows from several studies investigating Spatial frequency filters and also data integration for machine learning pipeline.

. Previous Works Supervised learning algorithms depend on the implicit assumption that training and testing data were both drawn from the same feature space and distribution. When this assumption is violated, there have traditionally been two approaches, both of which can be problematic: either fit an individual model for each domain, or ignore the differences in the data generating processes by learning a model on the pooled data [27]. A large amount of data may be required for the first approach, for the second, fitted models do not generalize to new domains. Newer transfer learning methods have shown more promising results in these circumstances. In particular, domain adaptation methods and more recently, domain generalization techniques, have received increasing attention. A number of approaches to domain generalization have been proposed. Complexity regularization approaches adapt the variability of the sampling distribution on tasks [20, 21, 22], including learning the sampling distribution directly [23]. A far more common approach is to learn invariant features among source domains, putting all tasks in a common feature space [24 - 28]. They include the Canonical Correlation Analysis (CCA) based approach proposed in [28], in which maximum mean discrepancy was used for domain distance regularization. In [26], the Domain Invariant Component Analysis (DICA) algorithm was used to learn an empirical mapping based on multi-domain data. Distribution mismatch across domains was minimized while the conditional functional relationships were preserved. The approach proposed by Motiian et al. [28] focused on minimizing the semantic alignment loss and the separation loss based on deep learning models, while that of Ghifary et al. [24] utilized a multi-task autoencoder to learn domain invariant features. Transfer Component Analysis (TCA) can been used to learn a shared subspace, minimizing dissimilarities across source domains, while maximally preserving data variance [25, 27]. Transfer components learned in a reproducing kernel Hilbert space (RKHS) form the shared subspace between source and target domains. To apply this for domain generalization, [29] derived the shared subspace between multiple source domains, and applied to related target domains. By assuming a Gaussian distribution for each dataset, a correspondence-free approach was adopted in [14] to align representations of MRI scans from three different datasets for classification problems on Alzheimer's disease (AD). Other applications in AD include [30] – [32]. Tumor and lesion classification have also received some attention [33]-[35]. To the best of our knowledge, this is the first time that dataset generalization has been applied to electroencephalographic data for the purpose of epileptic seizure detection.

Chapter 3

Methodology

3.1 Overview

The project work is divided into five stages: Architecture, description, Transformations, Collection and testing phase. The Architecture phase involves conceptualizing and designing steps and the various mathematical and semantic transformations that will be employed throughout the entire process. The Transformaations phase involves the implentation of several spatial, spectral, semantic and temporal transformations and pre-processing on the data. Collection phase is where we define the means by which we integrate the data and the testing phase involves testing the derived data against the individual datasets involved in the process on a pre-obatined deep neural network model.

3.2 Architecture

So far, the architecture is still being developed, fortunately it has been developed to level where we can draw out a block diagram to help guide the process. see Figure 3.1

Over the period of our investigation, the methodlogy evolves, with two major juctures with the later of the two being the utilised method. The initial method explored was the Normalization method and the second is the Supervised Domain Generation method.

3.2.1 Normalization method

This method although very naive seemed to be a great way to combine the chosen disparate EEG Datasets. It involves thinking of the makeup of the dataset as a set of 'filters' ,namely the: Spacial Filter, Spectral Filter and the Semantic Filter. The SSS filter method however has a fault in that for any new dataset to be combine the process would have to be done for all the dataset and would require an obscene amount of manpower and computation, this complexity becomes the prime con, inspiring the Final Hypothesis development, making onboarding new datasets and datapoints rather trivial and easy.

The SSS method requires that we list/characteriae all the properties for the datasets to be combined and then try to individually normalize properties that are not the same, prioritizing properties that tend to contribute the highest to the covariate shift between the datasets.

Spatial filter paramamters include: Electrode Quantity, Locations and Type, Montage. Spectral filter parameters include: Sampling rate, Preprocessing step. Lastly the Semantic filter parameters include the semantic annontations and storage format utilised. Other important parameters include the Record length etc So step-by-step anylysis would be:

1. Identify a list of parameters by which all existing seizure EEG datasets can be completely characterized.
2. Determine which parameters in (1) can be modified or transformed without negatively impacting the quality of data
3. Propose frameworks or techniques to transform each of the parameters in (2) into a consistent range.
4. Develop a computer application to implement the framework in (3) for at least two EEG seizure datasets.
5. Evaluate the performance of the application by carrying out an automatic seizure detection task on the combination of the datasets in (4).

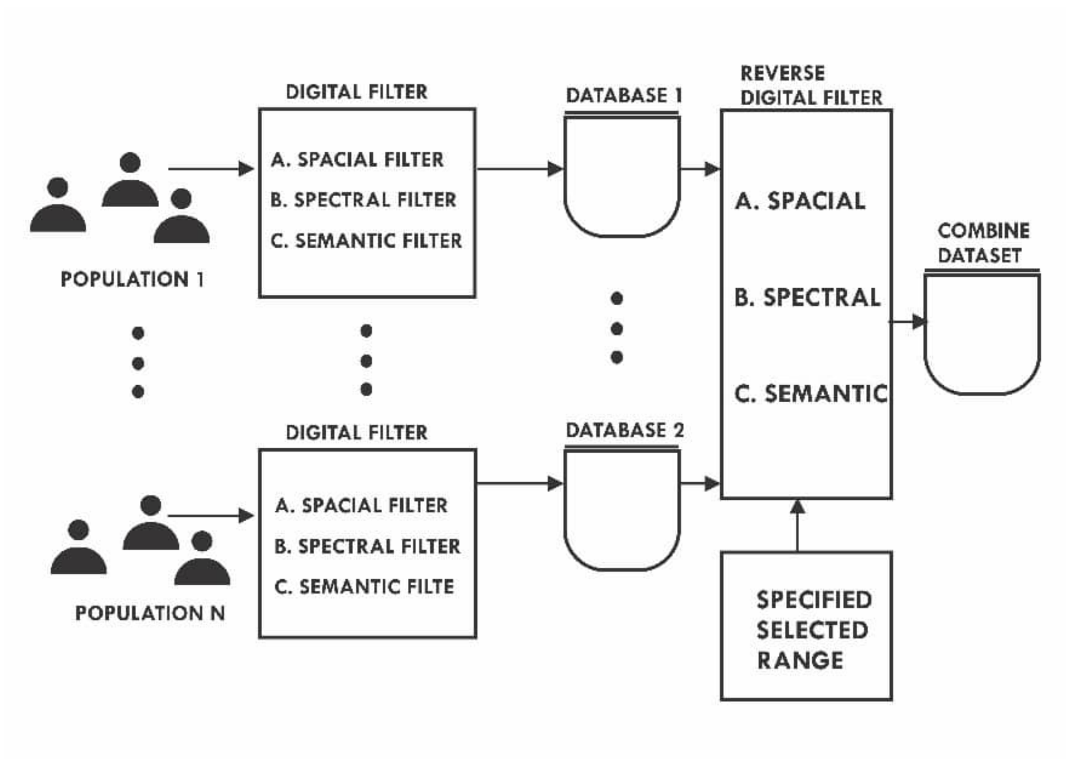

Figure 3.1: Block diagram for proposed technique (In a very broad-sense)

Bitmap Generation

One of the key techniques unique to our approach is the idea rasterization process or bitmap-generation. It is simply making images from EEG time-series data.

Electroencephalogram includes multiple time series corresponding to measurements across different spatial locations over the cortex. Similar to speech signals, the most salient features reside in frequency domain, usually studied using spectrogram of the signal. However, as already noted, EEG signal has an additional spatial dimension. Fast Fourier Transform (FFT) is performed on the time series for each trial to estimate the power spectrum of the signal. Oscillatory cortical activity related to memory operations primarily exists in three frequency bands of theta (4-7Hz), alpha (8-13Hz), and beta (13-30Hz) (Bashivan et al., 2014; Jensen & Tesche, 2002). Sum of squared absolute values within each of the three frequency bands was computed and used as separate measurement for each electrode. Aggregating spectral measurements for all electrodes to form a feature vector is the standard approach in EEG data analysis. However, this approach clearly ignores the inherent structure of the data in space, frequency, and time. Instead, we propose to transform the measurements into a 2-D image to preserve the spatial structure and use multiple color channels to represent the spectral di- mension. Finally, we use the sequence ofimages derived from consecutive time windows to account for temporal evolutions in brain activity. The EEG electrodes are distributed over the scalp in a three-dimensional space. In order to trans- form the spatially distributed activity maps as 2-D images, we need to first project the location of electrodes from a 3-dimensional space onto a 2-D surface. However, such transformation should also preserve the relative distance between neighboring electrodes. For this purpose, we used the Azimuthal Equidistant Projection (AEP) also known as Polar Projection, borrowed from mapping applications (Snyder, 1987). The azimuthal projections are formed onto a plane which is usually tangent to the globe at either pole, the Equator, or any intermediate point. In azimuthal equidistant projection, distances from the center of projection to any other point are preserved. Similarly, in our case the shape of the cap worn on a human's head can be approximated by a sphere and the same method could be used to compute the projection of electrode locations on a 2D surface that is tangent to the top point of the head. A drawback of this method is that the distances between the points on the map are only preserved with respect to a single point (the center point) and therefore the relative distances between all pairs of electrodes will not be exactly preserved. Applying AEP to 3-D electrode locations, we obtain 2-D projected locations of electrodes (Figure 3.2). Width and height of the image represent the spatial distribution of activities over the cortex. We apply Clough-Tocher scheme (Alfeld, 1984) for interpolating the scattered power measurements over the scalp and for estimating the values in-between the electrodes over a 32 x 32 mesh. This procedure is repeated for each frequency band of interest, resulting in three topographical activity maps corresponding to each frequency band. The three spatial maps are then merged together to form an image with three (color) channels. This three-channel image is given as an input to a deep convolutional network, as discussed in the following section. Figure 3.3 illustrates an overview of our multi-step approach to mental state classification from EEG data, where the novelty resides in transforming raw EEG into sequence of images, or frames (EEG "movie"), combined with recurrent-convolutional network architecture applied on top of such transformed EEG data. Note that our approach is general enough to be used in any EEG-based classification task, and a specific problem of mental load classification presented later only serves as an example demonstrating potential advantages of the proposed approach.

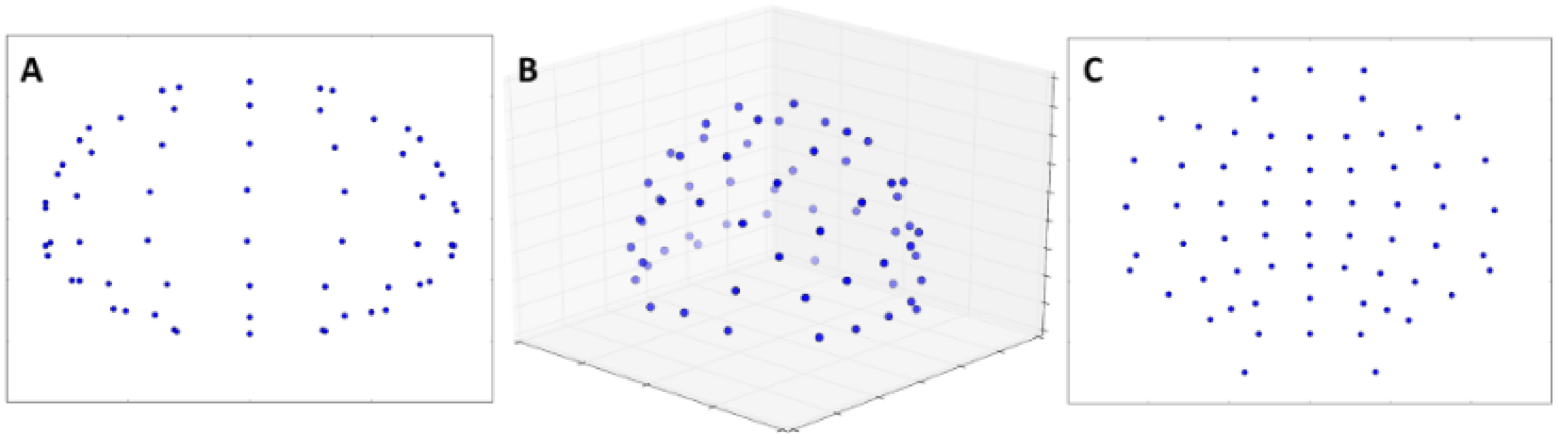

Figure 3.2: Topology-preserving and non-topology-preserving projections of electrode locations. A) 2-D projection of electrode locations using non-topology-preserving simple orthographic projection. B) Location of electrodes in the original 3-D space. C) 2-D projection of electrode locations using topology-preserving azimuthal equidistant projection.

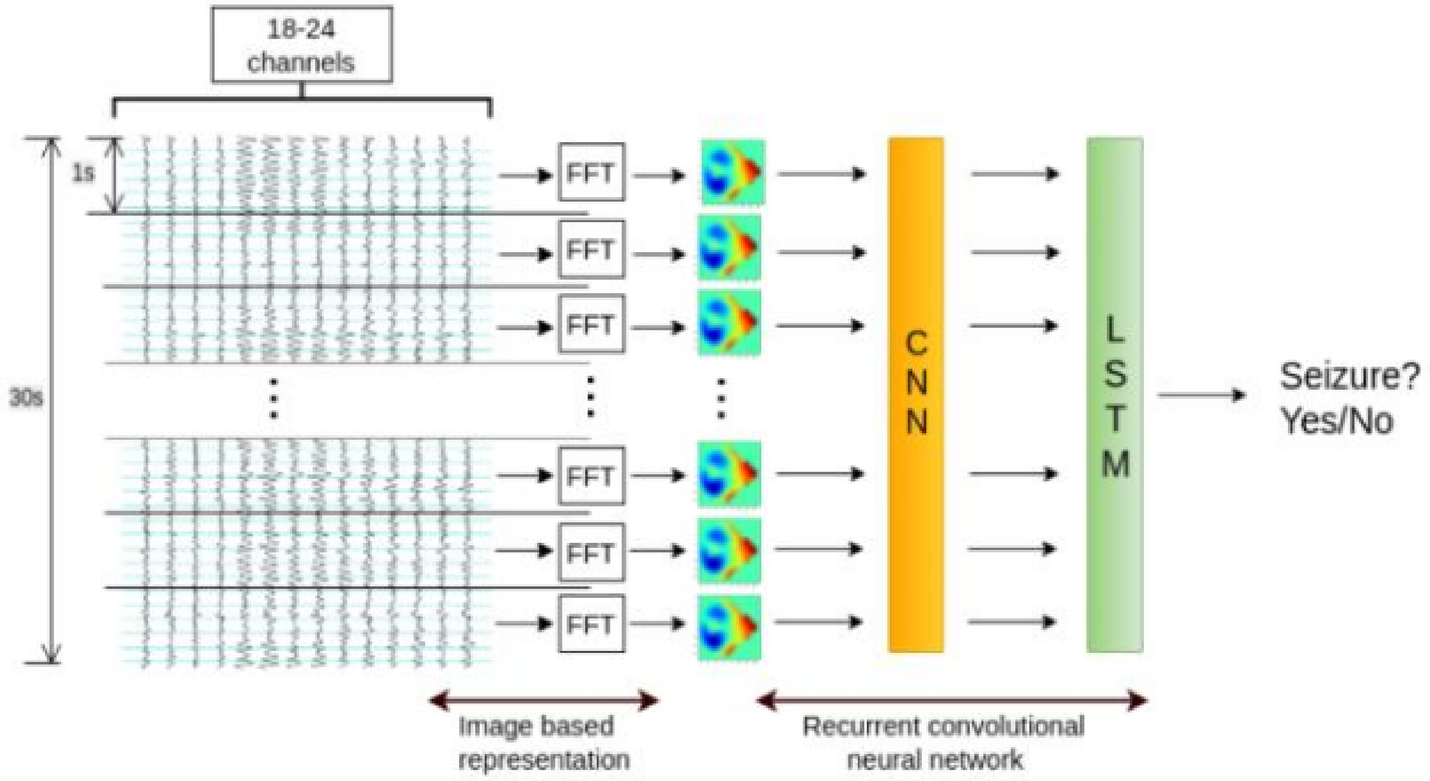
18-24 channels
1s
30s
FFT
C N N
L S T M
Seizure? Yes/No
Image based representation
Recurrent convolutional neural network

Figure 3.3: Diagrsm of a variant of recurrent convolutional neural network used for classification.

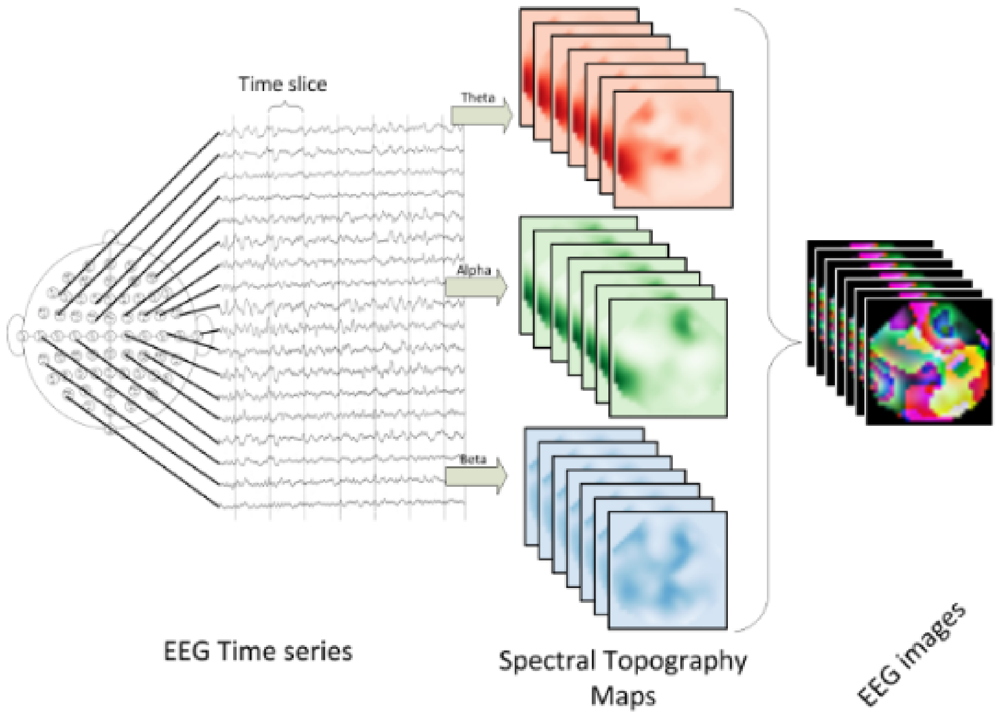

Figure 3.4: Overview of our approach: (1) EEG time series from multiple locations are acquired; (2) spectral power within three prominent frequency bands is extracted for each location and used to form topographical maps for each time frame (image); (3) sequence of topographical maps are combined to form a sequence of 3-channel images.

3.2.2 Supervised Domain Generation Method

A. Definitions

In this section, some important terms used throughout the paper are defined. This is all the more important because inconsistent notation and terminology is recognized as a problem in data quality research in general [39].

Definition 1 (Domain): A domain D is defined as $\mathbb{D} = \{X, P(x)\}$ where X is a feature space, $x \in X$, and P(x) is a marginal probability distribution over the feature space [37].

Definition 2 (Task): For a specific domain, a task T is defined as $\mathbb{T} = \{Y, P(y|x)\}$ where Y is a label space, $y \in Y$, and $P(y|x)$ is a conditional distribution over X Y [37].

Definition 3 (Dataset): A dataset D is defined as $D = \{X, Y, P(x), P(y|x)\}$, a set comprised of data belonging to a domain under a specific task [38].

Definition 4 (Supervised Domain Adaptation): Given two domains $\mathbb{D}^{(s)}$ and $\mathbb{D}^{(t)}$, where $\mathbb{D}^{(s)} \neq \mathbb{D}^{(t)}$, let samples $\mathbb{S}^{(s)} = \{x_i^{(s)}, y_i^{(s)}\}_{i=1}^{n_s}$ and $\mathbb{S}^{(t)} = \{x_i^{(t)}, y_i^{(t)}\}_{i=1}^{n_t}$ be drawn from the domains. The domain adaptation task is to learn a labeling function $f_{P(x)^t} : X \to Y$ given $\mathbb{S}^{(s)}$ and $\mathbb{S}^{(t)}$ as training examples [24].

Definition 5 (Supervised Domain Generalization): Given a set of m source domains $\mathbb{D} = \{\mathbb{D}^{(i)}\}_{i=1}^{m}$ and a target domain $\mathbb{D}^{(t)} \notin \mathbb{D}$, let samples $\mathbb{S}^{(s)} = \{\{x_i^{(j)}, y_i^{(j)}\}_{i=1}^{n_s}\}_{j=1}^{m}$ be drawn from the m source domains. The supervised domain generalization task is to learn a labelling function $f_{P(x)^t} : X \to Y$ given $\mathbb{S}^{(s)}$ as the training examples [24].

The similarities between supervised domain adaptation (SDA) and supervised domain generalization (SGD) should be evident from definitions 4 and 5. The goal in both cases is to learn a labelling function that performs well on a target domain. The primary difference lies in the need for additional training data to be drawn from the target domain in the case of SDA. The concepts of dataset shift in general, and covariate shift in particular are sources of a lot of the inconsistencies in notation and terminology in data quality research. For example, up to seven different uses of the concept of covariate shift were identified in [39]. The definition of covariate shift advanced in that study has been adopted for this work:

Definition 6 (Covariate Shift): Given two datasets $D^(s)$ and $D^(t)$, a covariate shift is said to have occurred if the marginal distribution changes, but the conditional distribution does not: $P^{(s)}(y|x) = P^{(t)}(y|x)$ but $P^{(s)}(x) \neq P^{(t)}(x)$ [39].

B. Problem Formulation

The typical strategy for automatic seizure detection is to train a classifier by minimizing some classification loss

$$\mathcal{L}_C(g) = E[\ell(g(x^{(s)}), y)] \tag{3.1}$$

where $E[]$ denotes statistical expectation, ℓ is an appropriate loss function [28], and the dataset from which $x^{(s)}$ and y are drawn represents an appropriate domain and task for epileptic seizure detection. Given K scalp EEG datasets, the total available training data can be represented thus:

$$D = \{\{x_i^{(j)}, y_i^{(j)}\}_{i=1}^{n_j}\}_{j=1}^{K} \tag{3.2}$$

where the feature $x^{(j)} \in \chi^{(j)}$is the i^{th} realization of an n- dimensional real-valued random variable $\mathbf{X^{(J)}} \in \mathbb{R}^\mathbf{n}$; $y_i^{(J)} \in Y$ is the label, and n^J is the number of samples in the J^{th} dataset.

The feature space $\chi^{(j)}$ for each dataset is derived from some underlying neurodynamic feature space Z^j through an EEG data acquisition and processing pipeline which can be represented by a mapping function:

$$f^{(J)}(\cdot) : Z^{(J)} \to \chi^{(J)} \tag{3.3}$$

Due to selection bias [11], $Z^{(J)}$ generally encloses only a subset of Z, the hypothetical set of all possible states of the unobserved neurodynamic variable z (see Figure 1). This is important because two implicit assumptions underlie the use or intended use of seizure detectors within a population:

$$P^{(j)}(z) = P^{(i)}(z) \qquad \forall\, i, j \tag{3.4}$$

$$P^{(j)}(x|z) = P^{(i)}(x|z) \qquad \forall\, i, j \tag{3.5}$$

For small datasets, the equality in eq. [4] will generally not hold true. If large datasets are used however, the likelihood is increased that a larger subset of Z is represented in the training set, making the assumption more accurate.

Combination of multiple independently-created datasets offers one possible route to increased training set sizes, especially if the records contained therein can be assumed to represent subjects from a human population with similar anatomical and physiological profiles. This route is however not usually explored for scalp EEG datasets because eq. (5) generally does not hold true. Even within the same population, humans exhibit wide variations in multiple anatomical and physiological traits. For example, cranium size and skull thickness vary greatly across individuals, and these, along with other factors, determine scalp potentials [40]. Consequently, given the same hypothetical unobserved cortical neurodynamic state, the potential gradient φ observable across the scalps of various subject will differ. Or,

$$P^{(i)}(\psi|z) \neq P^{(j)}(\psi|z) \qquad \forall \;\; i \neq j \tag{3.6}$$

Furthermore, given the exact same individual and same scalp potential gradient, different EEG acquisition pipelines will yield different dataset features x.

$$P^{(i)}(x^{(i)}|\psi) \neq P^{(j)}(x^{(j)}|\psi) \qquad \forall \;\; i \neq j \tag{3.7}$$

Where $x^{(i)} \in X^{(i)}$, $x^{(j)} \in X^{(j)}$ and the two feature spaces may be different: $X^{(i)} \neq X^{(j)}$. In the case where the two featurespaces are identical, and classifiers are to be trained on features drawn from the feature space, the problem in eq (7) is a covariate shift problem. Minimizing the distance between distributions provides a possible route to combining multiple disparate datasets. This is possible if some transformation, Γ is found so that:

$$P(\Gamma(x^{(i)}) = P(\Gamma(x^{(j)})) \tag{3.8}$$

If the total portion of the underlying neurodynamic space Z enclosed by the combined dataset is Z^T , and the classifier is to be deployed within a different population, this can be cast as a supervised domain generalization problem where the combined datasets form the source domain, and the target domain has features generated from Z^T.

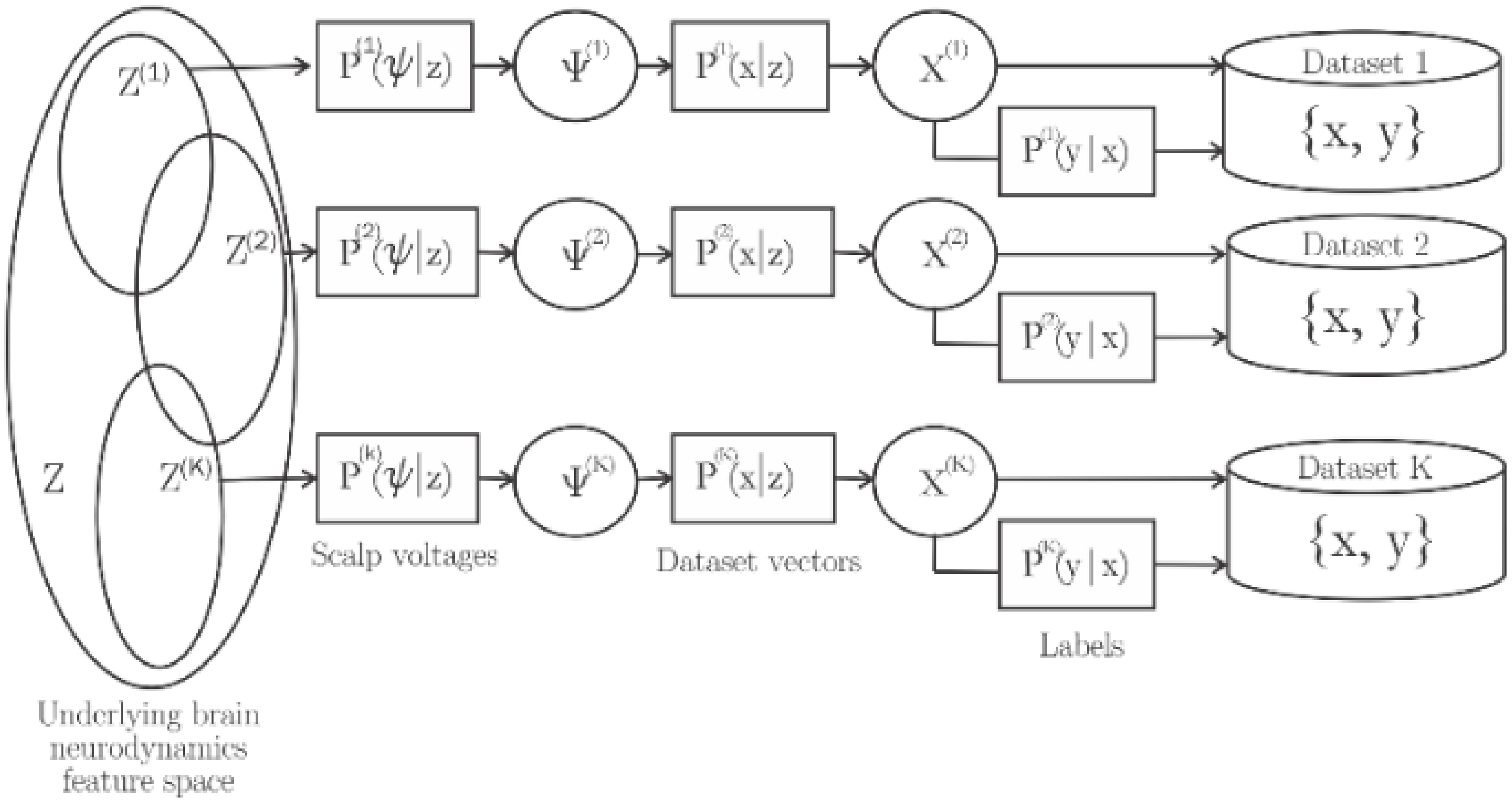

Figure 3.5: Acquisition and processing of scalp EEG data for multiple datasets, each from a subset of the same underlying neurodynamic feature space.

3.2.3 Experiment Setup

A. Datasets:

The only two large free and publicly-available scalp EEG datasets (CHB-MIT and TUSZ) constituted the source domains for this study. The need for a third scalp EEG dataset to use as the target domain posed a challenge as there is no other freely- available scalp EEG dataset (the Epilepsia dataset comes with a somewhat expensive license) and there was a reluctance to use an entirely proprietary dataset. In the end, it was decided to create a target dataset from the data of a number of randomly- selected CHB subject, a few randomly-selected TUSZ users, and a small 26-subject proprietary dataset ("OAUSZ1") acquired at the Obafemi Awolowo University Teaching Hospital Complex (OAUTHC), Nigeria. The OAUSZ1 dataset contained a total of 487 minutes of EEG records captured from 26 adult subjects in 27 sessions. All data were acquired in the neurology clinic of the OAUTHC using a Profusion Neuroscan EEG at a sampling rate of 256 Hz. The records contained a total of 10 seizures. Labelling of the data was done by three neurologists independently annotating the records. A simple voting scheme was used to determine seizure from the annotation of the neurologists, after which start and stop times were determined as averages of the independently indicated start and stop times. Acquisition of all data took place between January 2013 and February 2016. In order to maintain some balance in the source datasets, a selected subset of the TUSZ dataset was employed, which had approximately the same number of seizures as the CHB-MIT. A summary of the datasets used is presented in Table 2.

Dataset	Number of Subjects	Number of Seizures
CHB-MIT	23	198
TUSZ	29	201
OAUSZ-1	26	10
Total	78	409

Table 3.1: Summary of Dataset Information

B. Transformation into common feature space

For this study, it was assumed that each dataset feature vector was generated from a composite of instantaneous voltages measured simultaneously at multiple scalp locations, with respect to one or more references. Generally, scalp locations will vary, along with the montages, from dataset to dataset. Consequently, vectors from different scalp EEG datasets cannot be assumed to represent the same feature space. The CHB-MIT dataset for example uses variations on the double banana montage, while the TUSZ used both Temporal Central Parasagittal (TCP) and Average Reference (AR) montages.

Consistent use of the international 10-20 system of electrode placement allows 3-D coordinate information to be attached to each feature vector, providing a basis for transforming all source spaces into the same uniform geometrical space. Scalp EEG electrodes are located on the surface of a volume that can be modeled as a sphere with little loss of accuracy. It is advantageous to find a topology-preserving projection to transform them into 2-D space [10]. One such projection is a variant of polar projection [42] termed the Azimuthal Equidistant Projection (AEP), which preserves relative electrode distances to a single reference point, at the expense of slight errors in relative distance between any arbitrary set of electrodes [41]. The AEP was used to project an 18-electrode subset of the CHB feature vector space and a similar subset of the TUSZ space into onto a uniform plane constrained to 16x16 dimensions. Voltages at the 236 other locations within the plane were estimated by interpolation using the Clough-Tocher technique [43]. The entire process of transforming into a uniform vector space is depicted in Fig. 2. In the first step, the voltage simultaneously recorded at 18 electrode sites formed an 18x1 vector. This vector was transformed into an 18x4 vector by introducing x,y,z coordinates for each of the 18 original voltages. By subsequently carrying out AEP, the x,y,z coordinates for each electrode were reduced to x,y coordinates. Using these as the coordinates of 18 points on a plane, and the voltage of the signal acquired at that point as pixel intensity, a raster was created in which only 18-pixel locations are defined, as shown in Fig. 2, stage IV. All other pixel intensities were thereafter determined by interpolation (Stage V).

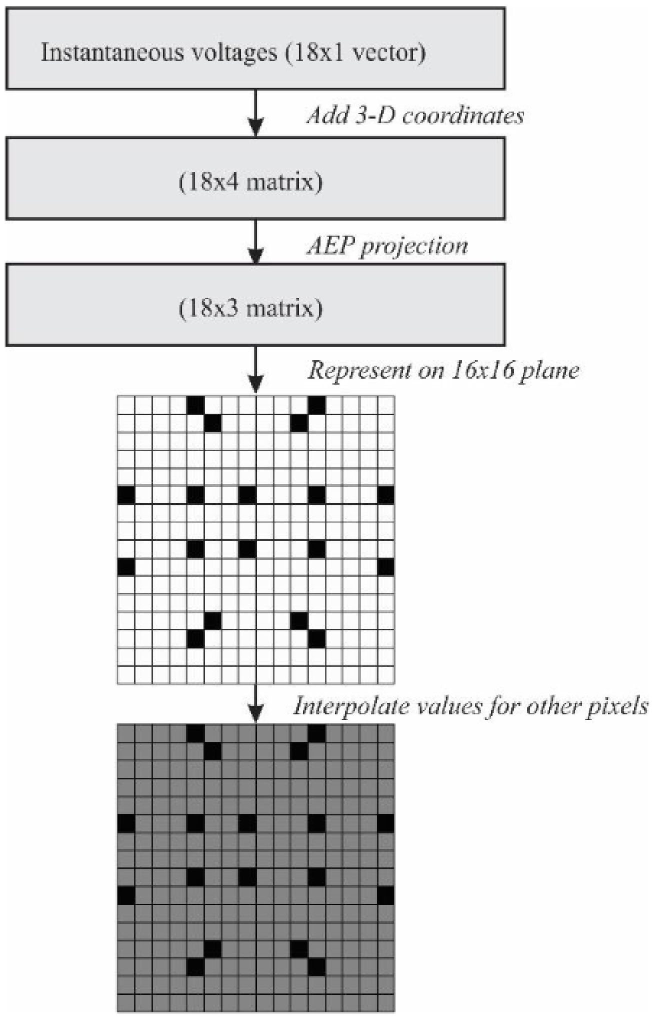

Figure 3.6: Transformation of an 18x1 vector onto a 2-D plane as a basis for projecting all vectors onto the same vector space.

C. Quantifying and Correcting Covariate Shift

To qualitatively assess the distributions of the source domains, histograms of the data distribution at 18 electrode locations in the CHB and TUSZ datasets were plotted. As seen from the corresponding distributions of data acquired from the Fp1-F7 electrode pair in Fig. 3, the distributions were all approximately Gaussian, although their leptokurtosis and higher peakedness, particularly for the TUSZ data, make them visually more reminiscent of the double exponential or hyperbolic secant distributions. Two-sample Kolmogorov-Smirnov tests supported the hypothesis at 5% significance that all CHB electrode data were drawn from the same distribution function, and all TUSZ electrode data were similarly from the same distribution function. TUSZ and CHB distributions were however different populations at 5% significance, which is evident in Fig. 3.

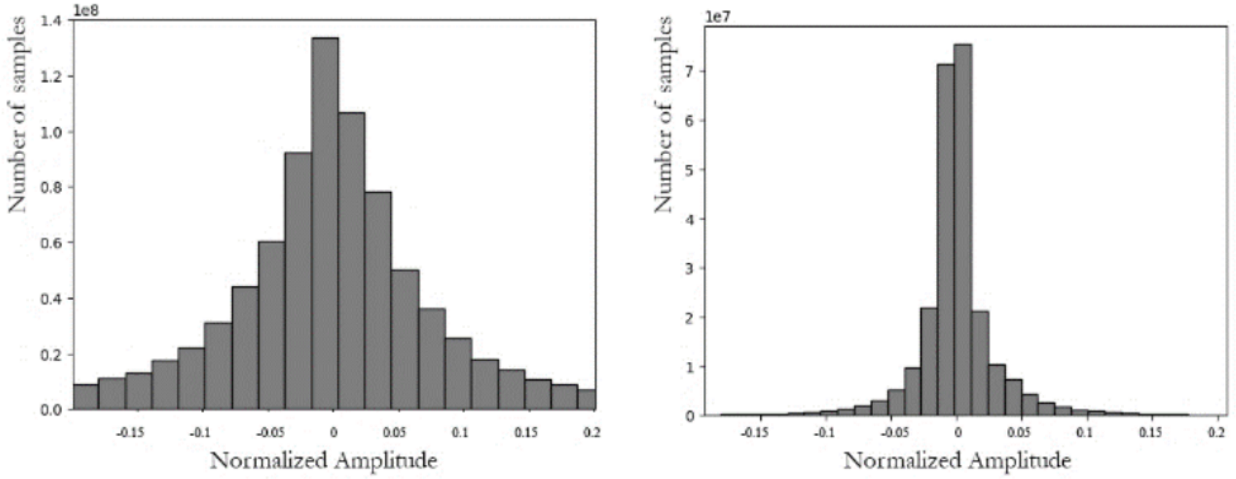

Figure 3.7: Distribution of data acquired at electrode Fp1-F7 for (a) CHB-MIT dataset (b)TUSZ dataset.

Most of the power in EEG signals are contained in the lower spectral components, and a distribution with a relatively high number of low-amplitudes components can be speculated to be one in which the higher amplitude (lower frequency) values are disproportionately attenuated. Consequently, an amplifier or filter with higher attenuation of low-frequency components could hypothetically result in a distribution with a narrower shoulder such as the TUSZ distribution. To explore this hypothesis, the difference in the frequency response of the TUSZ and CHB amplifiers was estimated by obtaining the average amplitude of each frequency bin after 256-point FFTs over both datasets. As seen in Fig 4, the TUSZ dataset does indeed exhibit consistently higher attenuation at lower frequencies, while higher frequency components are generally higher than the CHB equivalents. The spike at 60Hz for both datasets suggests significant contamination by the mains, despite filtering. By applying a filter with the approximate frequency response termed "average" in Fig. 4, the distributions of both datasets become more similar. Two-sample KS tests confirmed that the application of the filter made the two distributions much more visually similar, as can be observed in Fig. 5. A transfer learning approach based on a reproducing kernel Hilbert space (RKHS) was employed to learn a shared subspace in which the distance between the distributions is minimized, while preserving the statistical properties of the data. A modified version of the Multi-TCA algorithm [27], itself based on the TCA [25] was adopted. Assuming K source domains as before, the Multi-TCA algorithm finds the feature map, Γ, in equation (8) using the MMD-estimated measure of the distance between the means of distributions as a cost function. The MMD is given by

$$MMD = \frac{1}{K}\sum_{i=1}^{K} \|\mu_{x^{(i)}} - \mu_{\breve{x}}\|^2_{RKHS} \tag{3.9}$$

in which $\mu_{x^{(i)}} = \frac{1}{n^{(i)}}\sum_{j=1}^{n^{(i)}} \Gamma(x_j^{(i)})$, $\mu_{\breve{x}} = \frac{1}{K}\sum_{i=1}^{K} \mu_{x^{(i)}}$ and $\|\cdot\|$ RKHS is the RKHS norm.

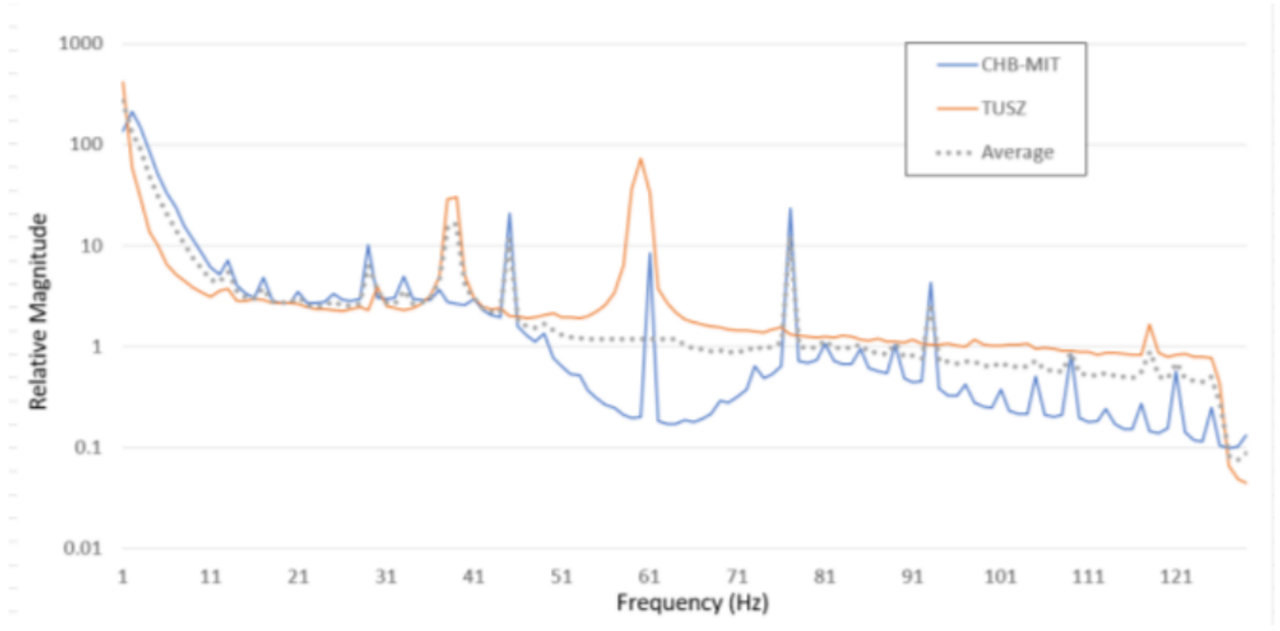

Figure 3.8: Estimated average power spectra across all electrodes of the TUSZ and CHB datasets.

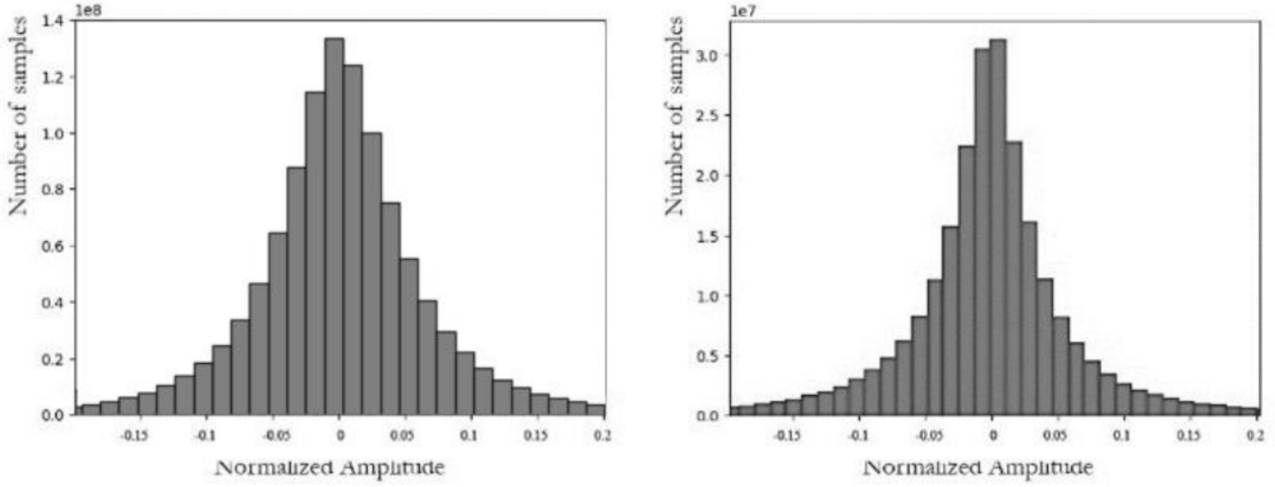

Figure 3.9: Distribution of data acquired at electrode Fp1-F7 data for (a) CHB-MIT dataset (b) filtered TUSZ dataset.

Given N total samples for which an m-dimensional embedding space is to be found, the MMD can be rapidly computed if a parametric kernel map $K = (KK^{-\frac{1}{2}})(K^{-\frac{1}{2}}K)$ is used, along with an orthogonal transformation matrix, $W \in \mathbb{R}^{Nxm}$, $m \ll N$ so that:

$$MMD = tr(W^T \; KLKW) \tag{3.10}$$

Data embedding into the latent space can subsequently be achieved by the transformation $W^T K$, and the solution of W is obtained by the m leading eigenvectors of $(KLK + \mu I)^{-1}KHK$ where μ is a parameter used to control the complexity of W. In this study, in order to preserve as much of the spatial and temporal information in the EEG data while carrying out transfer component analysis, the TCA-based dimensionality reduction was carried out on the spectral bins obtained for each epoch of data for each electrode. This was done by embedding the set of all 128-dimensional vectors acquired for each second at each site into a 10-dimensional space, as will be described in the next section.

D. Feature Vectors, Classifier Architecture and Training

One advantage of using the uniform vector space in Fig. 2 is that the feature vectors effectively become bitmaps, which brings with it the advantage that there are dozens of well-known architectures with excellent discriminatory performance with bitmaps. The conversion of feature vectors to bitmaps can be done in either time or frequency domains. A frequency domain approach was adopted for this study. Data continually sampled from each of 18 electrodes per session in a dataset were sliced into 256-sample epochs and each epoch transformed using the Fast Fourier Transform with a flattop window. The 128 components of the resulting spectrum were each transformed into a 10-dimensional embedding space by means of TCA. As shown in Figure 6, data from the corresponding component (of ten) from each electrode were combined to form an 18-bit vectors. Consequently, there were ten such vectors for each 1 s, 256-sample epoch. Each vector was projected onto a 16x16 grid using AEP, and the other pixels thereafter interpolated with Clough-Tocher technique. Consequently, each second of EEG record was converted into a ten-layered bitmap structure.

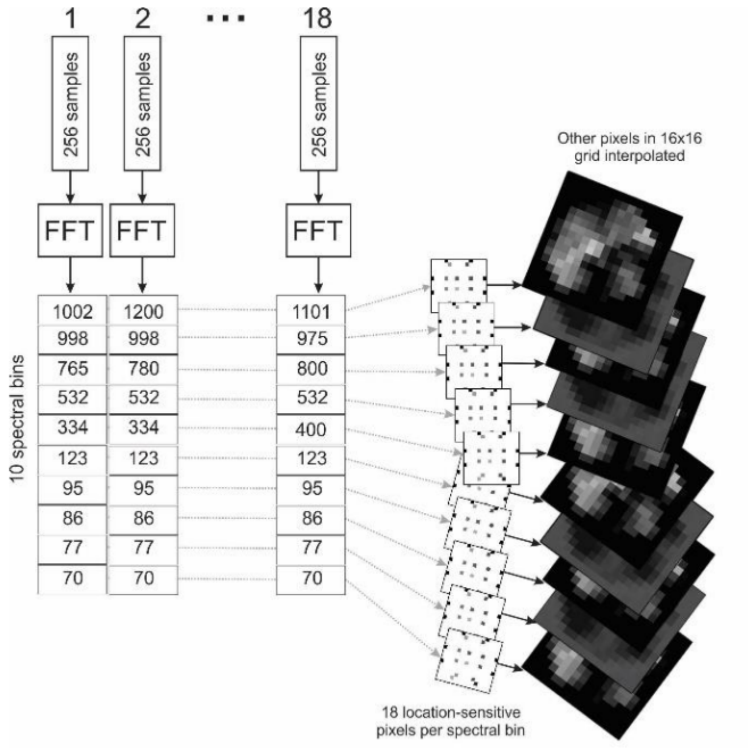

Figure 3.10: Generation of ten-layer bitmaps from 1-second-long portion of EEG records from 18 electrodes.

Epileptic seizure dynamics evolve over time, which suggests that feature vectors which encode some temporal information should be more desirable. As each ten-layer bitmap represents one second of EEG data, sequences of them were used as feature vectors for this study. When seizure detector features are sequences, the amount of time spanned by each vector is termed the frame length. A large variety of frame lengths have been employed for seizure detectors [44], ranging from 70 ms [45] to 1 hour [46]. For this work, a frame length of 5 seconds was used. This corresponded to a sequence of seventeen 1- second samples with 75% overlap. A recurrent convolutional neural network was used for classification. The architecture of the convolutional layers was inspired by the ConvNet Configuration D in [41], itself based on the VGG ImageNet classification network described in [47]. As shown in Fig. 7, the architecture had three stacks of four,two, and one convolutional layers respectively. The stacks were linked by 2x2 maxpool layers. All convolutional layers used a receptive field size of 3, a stride of 1 pixel, and padding of 1 to preserve spatial resolution; with 64, 128, and 256 filters respectively in the layers in each of the three stacks. The number of filters was selected based on the precept that output size is kept constant across the stacks of a VGG-style architecture by keeping the product of filter size and number of kernels constant. A long short-term memory (LSTM) network similar to [41] was adopted to decode temporal information], with the notable difference that in this study, 17 LSTM layers were stacked.

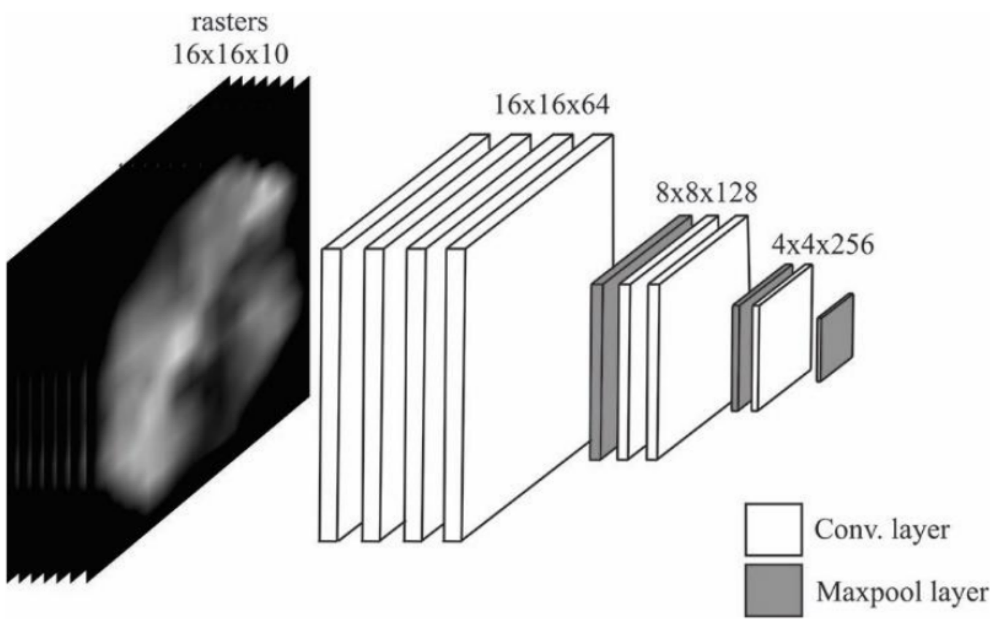

Figure 3.11: VGG-influenced convnet architecture.

Ideally, the combined CHB-MIT and TUSZ datasets would have been treated as the source domain, and a third dataset as the target domain. The relatively small size of the OAUSZ1 dataset was however problematic in this regard. It was decided to augment the size of the target dataset with selected data from the CHB-MIT and TUSZ. A four-fold cross-validation arrangement was set up, in which ultimately, a different portion of the CHB-MIT and TUSZ data were used as parts of the target datasets each time (as shown in Table 3), with the rest adopted as the source dataset. Training thereafter proceeded by using the source dataset for training, while the target dataset was used as the test dataset. Optimal parameters were obtained through a uniform sampling at random over hyper parameter space (see Table 4).

Fold	Subjects used as Target Domain (number of seizures in brackets)			
	CHB-MIT	TUSZ	OAUSZ1	Total
1.	1,2,3,4,5,6 (40)	1-8 (17)	1-26 (10)	42 (67)
2.	7-12 (62)	9-16 (22)	1-26 (10)	40 (94)
3.	13-18 (59)	17-24 (77)	1-26 (10)	40 (146)
4.	19-23 (37)	25-29 (85)	1-26 (10)	36 (132)

Table 3.2: Assignment of subjects to target domain for four folds

Hyperparameter	Value
Batch Size	64
Learning rate	0.001
Dropout probability	20%
Activation function	RELU
Number of epochs	50

Table 3.3: Summary of important hyperparameters

Chapter 4

Results and Discussion

4.1 Experimental Result

The research focused mainly on the use of domain generalization, which meant to minimize the covariate shift(marginal distribution changes, but the conditional distribution does not) and then training our classification on the new feature space obtained. The results are discussed in the sections to follow.

4.2 Convolution Operation to Reduce Covariate Shift

Lower frequencies hold most of the power in EEG signals, and a distribution with low amplitudes can be speculated to be one in which the higher amplitude (lower frequency) values are disproportionately attenuated. Consequently, an amplifier or filter with higher attenuation of low-frequency components could hypothetically result in a distribution with a narrower shoulder such as the TUSZ distribution. To explore this hypothesis, the difference in the frequency response of the TUSZ and CHB amplifiers was estimated by averaging a 256- point FFT over both datasets. As seen in Fig 4.2, the TUSZ dataset does indeed exhibit consistently higher attenuation at lower frequencies, while higher frequency components are generally higher than the CHB equivalents. The spike at 60Hz for both datasets suggests significant contamination by the mains despite filtering. By applying an appropriate filter to the TUSZ data to account for the frequency response disparity, the TUSZ frequency distribution became visually more similar to the CHB. Two-sample KS tests confirmed that the application of the filter made the two distributions essentially identical as can be observed in Figure 4.3. A combination of the two distributions was thereafter used to train a classifier as discussed in the next section.

The seizure detection accuracy per target dataset subset for the 4 folds is presented in Table 5. The average specificity for all datasets was 74.5%, while the false positive rates are summarized in Table 6. The average detection latency was 2.32s.

Fold	Target Domain			
	CHB-MIT Subset	TUSZ Subset	OAUSZ1	Combined
Fold 1	80.0	76.5	70.0	77.6
Fold 2	61.3	72.7	70.0	64.9
Fold 3	66.1	81.8	80.0	75.3
Fold 4	78.4	82.4	70.0	80.3
Average	71.45	78.35	72.5	74.5

Table 4.1: Summary of specificity for target datasets

Fold	Target Domain			
	CHB-MIT Subset	TUSZ Subset	OAUSZ1	Combined
Fold 1	0.75	1.14	0.61	0.83
Fold 2	0.82	0.87	0.74	0.81
Fold 3	0.78	0.76	0.75	0.76
Fold 4	0.77	0.82	1.33	0.97
Average	0.76	0.9	0.86	0.84

Table 4.2: Summary of false postives per hour

The performance of the classifier in this study is comparable to some previous results in studies that have used out-of-dataset data for validation, as shown in Table 7. In order to properly estimate how well a classifier might generalize with new populations, only studies

which used validation data drawn from populations partly or completely different from population used for classifier training are considered. The highest sensitivity was achieved in the Gabor study [8] in which 200 recordings from 65 patients was used to validate a previously-developed algorithm [48]. A sensitivity of 92.8% was achieved in the validation study. However, it is very important to note that although the patients in both training and validation studies were different, essentially the same acquisition pipelines were used for both, centering around a 64- channel BMSI 4000 data acquisition and review system, and an 8-channel Oxford Medilog 9000-II ambulatory recorder.
This amounted to ensuring that $P^{(}1)(x^{(}1)|\psi) = P^{(}2)(x^{(}2)|\psi)$ from equation (7), a situation which will not generally be possible for more general clinical deployment. Another large validation study by Hopfengärtner et al [9] achieved sensitivity of 87.3% on 159 patients, with a low FPH of 0.22 when validating an algorithm developed in [49]. This study used a different acquisition amplifier (64-channel NATUS amplifier) from the 64-channel IT-med system used for the acquisition of the original data with which the classifier was developed. However, it appears that deliberate efforts were made to minimize covariate shift between the training and validation datasets. For example, the same sampling rate and montage system were used. In addition, a low-pass filter with cutoff frequency of 0.08 Hz and a high pass filter with cutoff frequency of 86 Hz were applied to the data, to mirror the same filters used for the original study. Consequently, both the Gabor and Hopfengärtner studies achieved significantly higher performance than our results, but in both cases, the shift between training and validation datasets were minimized ab initio either by using the same acquisition pipeline hardware, or by conscious choice to adopt similar data processing settings. In contrast, the datasets used in the present study were acquired under completely different conditions. This suggests that our setup is more similar to what would occur in real-life clinical scenarios. In this regard, the REVEAL algorithm developed in [50] might be considered a much better simulation of the diversity of real-life data sources, and a better benchmark. The algorithm was developed using data acquired from multiple facilities, using different acquisition pipelines. One site adopted 8- channel, 8-bit Oxford Medilog 9000-II recorders sampling at 128 Hz, with filter settings of 0.5 – 40 Hz. A second site used 32-channel, 12-bit BMSI 5000 system sampled at 200 Hz, with filter setting of 0.1 to

70 Hz, while the third site used a 32- channel 12-bit Ceegraph amplifier sampled at 256 Hz with filter settings of 0.1 – 100 Hz. The only reported attempt to harmonize the data is that all datasets were reprocessed to use the same electrodes C3–F3, F3–F7, F7–T3, T3–T5, C4–F4, F4– F8, F8–T4, and T4–T6. Tested on this data, the REVEAL algorithm achieved an average sensitivity of 76%. When applied to the CHB-MIT dataset however, the sensitivity of REVEAL dropped to 67%. By way of comparison, the current algorithm achieved an average sensitivity of 71.45% on the CHB-MIT.

Study	Sensitivity	FPH
Gabor [8]	92.8	1.35
Hopfengärtner et al. [9]	87.3	0.22
REVEAL [50]	76	0.11
REVEAL with CHB-MIT	67	1.7
Current Study	74.5	0.76

Table 4.3: Comparism with existing validation studies

The goal of this study was not to insist that a superior performance to the state-of-the-art was achieved. In order to do that, much more efforts need to go towards ensuring more standardized comparisons. It suffices at the present to observe that the results suggest the viability of the approach proposed herein. That is, is possible to take data from completely different datasets, and combined them using transfer learning techniques, resulting in much larger datasets for training seizure detectors. A RKHS approach to transfer learning has been adopted in this study, but similar results could probably be obtained using other methods. Nor is this the only possible area to look towards for improvement. For example, the VGG architecture has been surpassed by both ResNet [51] and Inception [52] for image classification, and making appropriate modifications to the architecture of this study may result in improved results.

Chapter 5

Conclusion and Recommendation

5.1 Conclusion

From the results earlier shown and discussed in the previous, it can be seen that the aggregation of the disparate dataset by means of minimizing the covariance shift delivers results more accurate during inference and improves the generalization of the classifier model used due to the addition of more datapoints for training.

This study explored a domain generation approach to increasing the training dataset size for epileptic seizure detectors, by applying transfer component analysis to two disparate datasets, and using their combination to train a classifier. The results achieved compare well with results of previous large-scale validation studies. Although more work needs to be done to improve the technique, the results can at least be interpreted as an indication that if domain shift between disparate seizure EEG datasets can be minimized, they may be combined to increase the training dataset size available for automatic seizure detectors.

5.2 Recommendation

The results presented in this work are not technical and detailed enough. Further improvements need to be made in both the abstraction , number of modalities utilized and classifier architecture used. As regards the abstraction, this body of work shifts from the 'Filter Abstraction' to the 'Domain Generalization" as is explained in chapters 2 and 3, hence a better abstraction method could be devised to combine the datasets. As regards the number of modalites utilized, more neuroimaging modalities e.g MRI, MEG, fMRI, CAT can be used together with the EEG modality used to improve the techniques result. And lastly, a more neuroimaging specific representation learning architecture can be developed to accurately learn and generalize on the combined data.

Bibliography

Andrzejak, Ralph G, Kaspar Schindler, and Christian Rummel (2012). "Nonrandomness, nonlinear dependence, and nonstationarity of electroencephalographic recordings from epilepsy patients". In: *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics* 86.4. ISSN: 15393755. DOI: 10.1103/PhysRevE.86.046206.

Andrzejak, Ralph G et al. (2001). "Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity : Dependence on recording region and brain state". In: 64, pp. 1–8. DOI: 10.1103/PhysRevE.64.061907.

Bashivan, Pouya et al. (2015). "Learning Representations from EEG with Deep Recurrent-Convolutional Neural Networks". In: pp. 1–15. ISSN: 03610926. DOI: 10.1080/03610928808829796. arXiv: 1511.06448. URL: http://arxiv.org/abs/1511.06448.

Chen, Duo, Suiren Wan, and Forrest Sheng Bao (2016). "Epileptic Focus Localization Using Discrete Wavelet Transform Based on Interictal Intracranial EEG". In: 4320.c, pp. 1–12. DOI: 10.1109/TNSRE.2016.2604393.

Chen, Edward (2017). "YouTube-8M Video Understanding Challenge Approach and Applications". In: i. arXiv: 1706.08222. URL: http://arxiv.org/abs/1706.08222.

Gemmeke, Jort F. et al. (2017). "Audio Set: An ontology and human-labeled dataset for audio events". In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pp. 776–780. ISSN: 15206149. DOI: 10.1109/ICASSP.2017.7952261.

Goldberger, Ary L et al. (2000). "PhysioBank, PhysioToolkit, and PhysioNet Components of a New Research Resource for Complex Physiologic Signals Ary". In:

Gupta, Vipin et al. (2017). "Automated Detection of Focal EEG Signals using Features Extracted from Flexible Analytic Wavelet Transform". In: *Pattern Recognition Letters.*

ISSN: 0167-8655. DOI: 10.1016/j.patrec.2017.03.017. URL: http://dx.doi.org/10.1016/j.patrec.2017.03.017.

Hartmann, Kay Gregor, Robin Tibor Schirrmeister, and Tonio Ball (2018). "EEG-GAN: Generative adversarial networks for electroencephalograhic (EEG) brain signals". In: arXiv: 1806.01875. URL: http://arxiv.org/abs/1806.01875.

Itakura, Tatsunori, Toshihisa Tanaka, and A Dataset (2017). "Epileptic Focus Localization Based on Bivariate Empirical Mode Decomposition and Entropy". In: December, pp. 1426–1429.

Krell, Mario Michael and Su Kyoung Kim (2014). "Rotational Data Augmentation for Electroencephalographic Data". In: Section II.

Krizhevsky, Alex, Ilya Sutskever, and Hinton Geoffrey E. (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25 (NIPS2012)*, pp. 1–9. ISSN: 10495258. DOI: 10.1109/5.726791. arXiv: 1102.0183. URL: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Roy, Subhrajit, Isabell Kiral-kornek, and Stefan Harrer (2018). "ChronoNet : A Deep Recurrent Neural Network for Abnormal EEG Identification". In: *arXiv:1802.00308v2 [eess.SP]*, pp. 1–10. arXiv: arXiv:1802.00308v2.

Samie, Farzad, Sebastian Paul, and Lars Bauer (2018). "Highly Efficient and Accurate Seizure Prediction on Constrained IoT Devices". In: i, pp. 955–960.

Sanei, Saeid and Jonathon. A. Chambers (2007). *EEG Signal Processing.* Vol. 1. John Wiley & Sons Ltd, p. 289. ISBN: 9780470025819. DOI: 10.1002/9780470511923. arXiv: arXiv:1011.1669v3.

Shah, V et al. (2017). "OPTIMIZING CHANNEL SELECTION FOR SEIZURE DETECTION". In:

Shah, Vinit et al. (2018). "The Temple University Hospital Seizure Detection Corpus". In: 2.

Shoeb, Ali and John Guttag (2010). "Application of Machine Learning To Epileptic Seizure Detection". In:

Shoeb, Ali Hossam and Arthur C Smith (2004). “Patient-Specific Seizure Onset Detection”. In:

Sriraam, N and S Raghu (2017). “Classification of Focal and Non Focal Epileptic Seizures Using Multi-Features and SVM Classifier”. In: DOI: `10.1007/s10916-017-0800-x`.

Theodorakopoulou, Andriana (2017). “Machine learning data preparation for epileptic seizures prediction”. In: June.

Thodoroff, Pierre, Joelle Pineau, and Andrew Lim (2016). “Learning Robust Features using Deep Learning for Automatic Seizure Detection”. In: 56.

Truong, Nhan Duy et al. (2018). “Convolutional neural networks for seizure prediction using intracranial and scalp electroencephalogram Convolutional Neural Networks for Seizure Prediction Using Intracranial and Scalp Electroencephalogram”. In: *Neural Networks*. ISSN: 0893-6080. DOI: `10.1016/j.neunet.2018.04.018`. URL: `https://doi.org/10.1016/j.neunet.2018.04.018`.

Ullah, Ihsan et al. (2010). “An Automated System for Epilepsy Detection using EEG Brain Signals based on Deep Learning Approach Insight Centre for Data Analytics , National University of Ireland , Galway , Ireland Visual Computing Lab , Department of Computer Science , College of Com”. In: Ml.

Wang, Jason and Luis Perez (2017). “The Effectiveness of Data Augmentation in Image Classification using Deep Learning”. In: *Unpublished*. arXiv: `1712.04621`. URL: `http://cs231n.stanford.edu/reports/2017/pdfs/300.pdf{\%}0Ahttp://arxiv.org/abs/1712.04621`.

Weltin, E Von et al. (2018). “Electroencephalographic Slowing: A Primary Source of Error in Automatic Seizure Detection”. In:

Ziyabari, Saeedeh, Vinit Shah, and Meysam Golmohammadi (2017). “Objective evaluation metrics for automatic classification of EEG events”. In: pp. 1–17.

Appendix A

Utils.py

```
__author__ = 'Wisdom Ikezogwo'
#from __future__ import print_function
import math as m
import numpy as np
np.random.seed(123)
import scipy.io
from sklearn.decomposition import PCA

def cart2sph(x, y, z):
    """
    Transform Cartesian coordinates to spherical
    :param x: X coordinate
    :param y: Y coordinate
    :param z: Z coordinate
    :return: radius, elevation, azimuth
    """
    x2_y2 = x**2 + y**2
    r = m.sqrt(x2_y2 + z**2) # r
    elev = m.atan2(z, m.sqrt(x2_y2)) # Elevation
    az = m.atan2(y, x) # Azimuth
    return r, elev, az


def pol2cart(theta, rho):
    """
    Transform polar coordinates to Cartesian
    :param theta: angle value
    :param rho: radius value
    :return: X, Y
    """
    return rho * m.cos(theta), rho * m.sin(theta)


def augment_EEG(data, stdMult, pca=False, n_components=2):
    """
    Augment data by adding normal noise to each feature.

    :param data: EEG feature data as a matrix (n_samples x n_features)
    :param stdMult: Multiplier for std of added noise
    :param pca: if True will perform PCA on data and add noise proportional to PCA components.
    :param n_components: Number of components to consider when using PCA.
    :return: Augmented data as a matrix (n_samples x n_features)
    """
    augData = np.zeros(data.shape)
    if pca:
        pca = PCA(n_components=n_components)
        pca.fit(data)
        components = pca.components_
        variances = pca.explained_variance_ratio_
        coeffs = np.random.normal(scale=stdMult, size=pca.n_components) * variances
        for s, sample in enumerate(data):
            augData[s, :] = sample + (components * coeffs.reshape((n_components, -1))).sum(axis=0)
    else:
        # Add Gaussian noise with std determined by weighted std of each feature
        for f, feat in enumerate(data.transpose()):
            augData[:, f] = feat + np.random.normal(scale=stdMult*np.std(feat), size=feat.size)
    return augData


def augment_EEG_image(image, std_mult, pca=False, n_components=2):
    """
    Augment data by adding normal noise to each feature.
    :param image: EEG feature data as a a colored image [n_samples, n_colors, W, H]
    :param std_mult: Multiplier for std of added noise
    :param pca: if True will perform PCA on data and add noise proportional to PCA components.
    :param n_components: Number of components to consider when using PCA.
    :return: Augmented data as a matrix (n_samples x n_features)
    """
    augData = np.zeros((data.shape[0], data.shape[1], data.shape[2] * data.shape[3]))
    for c in range(image.shape[1]):
        reshData = np.reshape(data['featMat'][:, c, :, :], (data['featMat'].shape[0], -1))
        if pca:
            augData[:, c, :] = augment_EEG(reshData, std_mult, pca=True, n_components=n_components
                )
        else:
            augData[:, c, :] = augment_EEG(reshData, std_mult, pca=False)
    return np.reshape(augData, data['featMat'].shape)


def load_data(data_file):
    """
    Loads the data from MAT file. MAT file should contain two
    variables. 'featMat' which contains the feature matrix in the
    shape of [samples, features] and 'labels' which contains the output
    labels as a vector. Label numbers are assumed to start from 1.

    Parameters
    ----------
    data_file: str

    Returns
    -------
    data: array_like
    """
    print("Loading data from %s" % (data_file))

    dataMat = scipy.io.loadmat(data_file, mat_dtype=True)
    print(”Data loading complete. Shape is %r” % (dataMat[’featMat’].shape,))
    return dataMat[’features’][:, :−1], dataMat[’features’][:, −1] − 1 # Sequential indices


def reformatInput(data, labels, indices):
    ”””
    Receives the the indices for train and test datasets.
    Outputs the train, validation, and test data and label datasets.
    ”””

    trainIndices = indices[0][len(indices[1]):]
    validIndices = indices[0][:len(indices[1])]
    testIndices = indices[1]
    # Shuffling training data
    # shuffledIndices = np.random.permutation(len(trainIndices))
    # trainIndices = trainIndices[shuffledIndices]
    if data.ndim == 4:
        return [(data[trainIndices], np.squeeze(labels[trainIndices]).astype(np.int32)),
                (data[validIndices], np.squeeze(labels[validIndices]).astype(np.int32)),
                (data[testIndices], np.squeeze(labels[testIndices]).astype(np.int32))]
    elif data.ndim == 5:
        return [(data[:, trainIndices], np.squeeze(labels[trainIndices]).astype(np.int32)),
                (data[:, validIndices], np.squeeze(labels[validIndices]).astype(np.int32)),
                (data[:, testIndices], np.squeeze(labels[testIndices]).astype(np.int32))]


if __name__ == ’__main__’:
    data = np.random.normal(size=(100, 10))
    print (’Original: {0}’.format(data))
    print (’Augmented: {0}’.format(augment_EEG(data, 0.1, pca=True)))
```

EEG_to_image_sequence.py

```
from __future__ import print_function
import time


import numpy as np
```

```
np.random.seed(1234)
from functools import reduce
import math as m

import scipy.io
import theano
import theano.tensor as T

from scipy.interpolate import griddata
from sklearn.preprocessing import scale
from utils import augment_EEG, cart2sph, pol2cart

import lasagne
# from lasagne.layers.dnn import Conv2DDNNLayer as ConvLayer
from lasagne.layers import Conv2DLayer, MaxPool2DLayer, InputLayer
from lasagne.layers import DenseLayer, ElemwiseMergeLayer, FlattenLayer
from lasagne.layers import ConcatLayer, ReshapeLayer, get_output_shape
from lasagne.layers import Conv1DLayer, DimshuffleLayer, LSTMLayer, SliceLayer


def azim_proj(pos):
    """
    Computes the Azimuthal Equidistant Projection of input point in 3D Cartesian Coordinates.
    Imagine a plane being placed against (tangent to) a globe. If
    a light source inside the globe projects the graticule onto
    the plane the result would be a planar, or azimuthal, map
    projection.

    :param pos: position in 3D Cartesian coordinates
    :return: projected coordinates using Azimuthal Equidistant Projection
    """
    [r, elev, az] = cart2sph(pos[0], pos[1], pos[2])
    return pol2cart(az, m.pi / 2 - elev)


def gen_images(locs, features, n_gridpoints, normalize=True,
               augment=False, pca=False, std_mult=0.1, n_components=2, edgeless=False):
```

```
"""
Generates EEG images given electrode locations in 2D space and multiple feature values for each
    electrode

:param locs: An array with shape [n_electrodes, 2] containing X, Y
                coordinates for each electrode.
:param features: Feature matrix as [n_samples, n_features]
                        Features are as columns.
                        Features corresponding to each frequency band are concatenated.
                        (alpha1, alpha2, ..., beta1, beta2,...)
:param n_gridpoints: Number of pixels in the output images
:param normalize: Flag for whether to normalize each band over all samples
:param augment: Flag for generating augmented images
:param pca: Flag for PCA based data augmentation
:param std_mult Multiplier for std of added noise
:param n_components: Number of components in PCA to retain for augmentation
:param edgeless: If True generates edgeless images by adding artificial channels
                at four corners of the image with value = 0 (default=False).
:return: Tensor of size [samples, colors, W, H] containing generated
                images.
"""
feat_array_temp = []
nElectrodes = locs.shape[0] # Number of electrodes
# Test whether the feature vector length is divisible by number of electrodes
assert features.shape[1] % nElectrodes == 0
n_colors = features.shape[1] / nElectrodes
for c in range(n_colors):
    feat_array_temp.append(features[:, c * nElectrodes : nElectrodes * (c+1)])
if augment:
    if pca:
        for c in range(n_colors):
            feat_array_temp[c] = augment_EEG(feat_array_temp[c], std_mult, pca=True,
                n_components=n_components)
    else:
        for c in range(n_colors):
            feat_array_temp[c] = augment_EEG(feat_array_temp[c], std_mult, pca=False,
                n_components=n_components)
    nSamples = features.shape[0]
    # Interpolate the values
    grid_x, grid_y = np.mgrid[
                     min(locs[:, 0]):max(locs[:, 0]):n_gridpoints*1j,
                     min(locs[:, 1]):max(locs[:, 1]):n_gridpoints*1j
                     ]
    temp_interp = []
    for c in range(n_colors):
        temp_interp.append(np.zeros([nSamples, n_gridpoints, n_gridpoints]))
    # Generate edgeless images
    if edgeless:
        min_x, min_y = np.min(locs, axis=0)
        max_x, max_y = np.max(locs, axis=0)
        locs = np.append(locs, np.array([[min_x, min_y], [min_x, max_y],[max_x, min_y],[max_x, max_y]]),
             axis=0)
        for c in range(n_colors):
            feat_array_temp[c] = np.append(feat_array_temp[c], np.zeros((nSamples, 4)), axis=1)
    # Interpolating
    for i in xrange(nSamples):
        for c in range(n_colors):
            temp_interp[c][i, :, :] = griddata(locs, feat_array_temp[c][i, :], (grid_x, grid_y),
                                    method='cubic', fill_value=np.nan)
        print('Interpolating {0}/{1}\r'.format(i+1, nSamples), end='\r')
    # Normalizing
    for c in range(n_colors):
        if normalize:
            temp_interp[c][~np.isnan(temp_interp[c])] = \
                scale(temp_interp[c][~np.isnan(temp_interp[c])])
        temp_interp[c] = np.nan_to_num(temp_interp[c])
    return np.swapaxes(np.asarray(temp_interp), 0, 1) # swap axes to have [samples, colors, W, H]


def build_cnn(input_var=None, w_init=None, n_layers=(4, 2, 1), n_filters_first=32, imsize=32, n_colors=3):
    """
    Builds a VGG style CNN network followed by a fully−connected layer and a softmax layer.
    Stacks are separated by a maxpool layer. Number of kernels in each layer is twice
    the number in previous stack.
    input_var: Theano variable for input to the network
    outputs: pointer to the output of the last layer of network (softmax)

    :param input_var: theano variable as input to the network
    :param w_init: Initial weight values
    :param n_layers: number of layers in each stack. An array of integers with each
                    value corresponding to the number of layers in each stack.
                    (e.g. [4, 2, 1] == 3 stacks with 4, 2, and 1 layers in each.
    :param n_filters_first: number of filters in the first layer
    :param imSize: Size of the image
    :param n_colors: Number of color channels (depth)
    :return: a pointer to the output of last layer
    """
    weights = [] # Keeps the weights for all layers
    count = 0
    # If no initial weight is given, initialize with GlorotUniform
    if w_init is None:
        w_init = [lasagne.init.GlorotUniform()] * sum(n_layers)
    # Input layer
    network = InputLayer(shape=(None, n_colors, imsize, imsize),
                                        input_var=input_var)
    for i, s in enumerate(n_layers):
        for l in range(s):
            network = Conv2DLayer(network, num_filters=n_filters_first * (2 ** i), filter_size=(3, 3),
                          W=w_init[count], pad='same')
            count += 1
            weights.append(network.W)
        network = MaxPool2DLayer(network, pool_size=(2, 2))
    return network, weights


def build_convpool_max(input_vars, nb_classes, imsize=32, n_colors=3, n_timewin=3):
    """
    Builds the complete network with maxpooling layer in time.

    :param input_vars: list of EEG images (one image per time window)
    :param nb_classes: number of classes
    :param imsize: size of the input image (assumes a square input)
    :param n_colors: number of color channels in the image
    :param n_timewin: number of time windows in the snippet
    :return: a pointer to the output of last layer
    """
    convnets = []
    w_init = None
    # Build 7 parallel CNNs with shared weights
    for i in range(n_timewin):
        if i == 0:
            convnet, w_init = build_cnn(input_vars[i], imsize=imsize, n_colors=n_colors)
        else:
            convnet, _ = build_cnn(input_vars[i], w_init=w_init, imsize=imsize, n_colors=n_colors)
        convnets.append(convnet)
    # convpooling using Max pooling over frames
    convpool = ElemwiseMergeLayer(convnets, theano.tensor.maximum)
    # A fully-connected layer of 512 units with 50% dropout on its inputs:
    convpool = DenseLayer(lasagne.layers.dropout(convpool, p=.5),
            num_units=512, nonlinearity=lasagne.nonlinearities.rectify)
    # And, finally, the output layer with 50% dropout on its inputs:
    convpool = lasagne.layers.DenseLayer(lasagne.layers.dropout(convpool, p=.5),
            num_units=nb_classes, nonlinearity=lasagne.nonlinearities.softmax)
    return convpool


def build_convpool_conv1d(input_vars, nb_classes, imsize=32, n_colors=3, n_timewin=3):
    """
    Builds the complete network with 1D-conv layer to integrate time from sequences of EEG images.

    :param input_vars: list of EEG images (one image per time window)
    :param nb_classes: number of classes
    :param imsize: size of the input image (assumes a square input)
    :param n_colors: number of color channels in the image
    :param n_timewin: number of time windows in the snippet
    :return: a pointer to the output of last layer
    """
    convnets = []
    w_init = None
    # Build 7 parallel CNNs with shared weights
    for i in range(n_timewin):
        if i == 0:
            convnet, w_init = build_cnn(input_vars[i], imsize=imsize, n_colors=n_colors)
        else:
            convnet, _ = build_cnn(input_vars[i], w_init=w_init, imsize=imsize, n_colors=n_colors)
        convnets.append(FlattenLayer(convnet))
    # at this point convnets shape is [numTimeWin][n_samples, features]
    # we want the shape to be [n_samples, features, numTimeWin]
    convpool = ConcatLayer(convnets)
    convpool = ReshapeLayer(convpool, ([0], n_timewin, get_output_shape(convnets[0])[1]))
    convpool = DimshuffleLayer(convpool, (0, 2, 1))
    # input to 1D convlayer should be in (batch_size, num_input_channels, input_length)
    convpool = Conv1DLayer(convpool, 64, 3)
    # A fully−connected layer of 512 units with 50% dropout on its inputs:
    convpool = DenseLayer(lasagne.layers.dropout(convpool, p=.5),
            num_units=512, nonlinearity=lasagne.nonlinearities.rectify)
    # And, finally, the output layer with 50% dropout on its inputs:
    convpool = DenseLayer(lasagne.layers.dropout(convpool, p=.5),
            num_units=nb_classes, nonlinearity=lasagne.nonlinearities.softmax)
    return convpool


def build_convpool_lstm(input_vars, nb_classes, grad_clip=110, imsize=32, n_colors=3, n_timewin=3):
    """
    Builds the complete network with LSTM layer to integrate time from sequences of EEG images.

    :param input_vars: list of EEG images (one image per time window)
    :param nb_classes: number of classes
    :param grad_clip: the gradient messages are clipped to the given value during
                      the backward pass.
    :param imsize: size of the input image (assumes a square input)
    :param n_colors: number of color channels in the image
    :param n_timewin: number of time windows in the snippet
    :return: a pointer to the output of last layer
    """
    convnets = []
    w_init = None
    # Build 7 parallel CNNs with shared weights
    for i in range(n_timewin):
        if i == 0:
            convnet, w_init = build_cnn(input_vars[i], imsize=imsize, n_colors=n_colors)
        else:
            convnet, _ = build_cnn(input_vars[i], w_init=w_init, imsize=imsize, n_colors=n_colors)
        convnets.append(FlattenLayer(convnet))
    # at this point convnets shape is [numTimeWin][n_samples, features]
    # we want the shape to be [n_samples, features, numTimeWin]
    convpool = ConcatLayer(convnets)
    convpool = ReshapeLayer(convpool, ([0], n_timewin, get_output_shape(convnets[0])[1]))
    # Input to LSTM should have the shape as (batch size, SEQ_LENGTH, num_features)
    convpool = LSTMLayer(convpool, num_units=128, grad_clipping=grad_clip,
        nonlinearity=lasagne.nonlinearities.tanh)
    # We only need the final prediction, we isolate that quantity and feed it
    # to the next layer.
    convpool = SliceLayer(convpool, −1, 1) # Selecting the last prediction
    # A fully−connected layer of 256 units with 50% dropout on its inputs:
    convpool = DenseLayer(lasagne.layers.dropout(convpool, p=.5),
            num_units=256, nonlinearity=lasagne.nonlinearities.rectify)
    # And, finally, the output layer with 50% dropout on its inputs:
    convpool = DenseLayer(lasagne.layers.dropout(convpool, p=.5),
            num_units=nb_classes, nonlinearity=lasagne.nonlinearities.softmax)
    return convpool


def build_convpool_mix(input_vars, nb_classes, grad_clip=110, imsize=32, n_colors=3, n_timewin=3):
    ”””
    Builds the complete network with LSTM and 1D−conv layers combined


    :param input_vars: list of EEG images (one image per time window)
    :param nb_classes: number of classes
    :param grad_clip: the gradient messages are clipped to the given value during
                        the backward pass.
    :param imsize: size of the input image (assumes a square input)
    :param n_colors: number of color channels in the image
    :param n_timewin: number of time windows in the snippet
    :return: a pointer to the output of last layer
    """
    convnets = []
    w_init = None
    # Build 7 parallel CNNs with shared weights
    for i in range(n_timewin):
        if i == 0:
            convnet, w_init = build_cnn(input_vars[i], imsize=imsize, n_colors=n_colors)
        else:
            convnet, _ = build_cnn(input_vars[i], w_init=w_init, imsize=imsize, n_colors=n_colors)
        convnets.append(FlattenLayer(convnet))
    # at this point convnets shape is [numTimeWin][n_samples, features]
    # we want the shape to be [n_samples, features, numTimeWin]
    convpool = ConcatLayer(convnets)
    convpool = ReshapeLayer(convpool, ([0], n_timewin, get_output_shape(convnets[0])[1]))
    reformConvpool = DimshuffleLayer(convpool, (0, 2, 1))
    # input to 1D convlayer should be in (batch_size, num_input_channels, input_length)
    conv_out = Conv1DLayer(reformConvpool, 64, 3)
    conv_out = FlattenLayer(conv_out)
    # Input to LSTM should have the shape as (batch size, SEQ_LENGTH, num_features)
    lstm = LSTMLayer(convpool, num_units=128, grad_clipping=grad_clip,
        nonlinearity=lasagne.nonlinearities.tanh)
    lstm_out = SliceLayer(lstm, -1, 1)
    # Merge 1D-Conv and LSTM outputs
    dense_input = ConcatLayer([conv_out, lstm_out])
    # A fully-connected layer of 256 units with 50% dropout on its inputs:
    convpool = DenseLayer(lasagne.layers.dropout(dense_input, p=.5),
            num_units=512, nonlinearity=lasagne.nonlinearities.rectify)
    # And, finally, the 10-unit output layer with 50% dropout on its inputs:
    convpool = DenseLayer(convpool,
            num_units=nb_classes, nonlinearity=lasagne.nonlinearities.softmax)
    return convpool


def iterate_minibatches(inputs, targets, batchsize, shuffle=False):
    """
    Iterates over the samples returing batches of size batchsize.
    :param inputs: input data array. It should be a 4D numpy array for images [n_samples, n_colors, W, H
        ] and 5D numpy
                    array if working with sequence of images [n_timewindows, n_samples, n_colors, W, H].
    :param targets: vector of target labels.
    :param batchsize: Batch size
    :param shuffle: Flag whether to shuffle the samples before iterating or not.
    :return: images and labels for a batch
    """
    if inputs.ndim == 4:
        input_len = inputs.shape[0]
    elif inputs.ndim == 5:
        input_len = inputs.shape[1]
    assert input_len == len(targets)
    if shuffle:
        indices = np.arange(input_len)
        np.random.shuffle(indices)
    for start_idx in range(0, input_len, batchsize):
        if shuffle:
            excerpt = indices[start_idx:start_idx + batchsize]
        else:
            excerpt = slice(start_idx, start_idx + batchsize)
        if inputs.ndim == 4:
            yield inputs[excerpt], targets[excerpt]
        elif inputs.ndim == 5:
            yield inputs[:, excerpt], targets[excerpt]


def train(images, labels, fold, model_type, batch_size=32, num_epochs=5):
    """
    A sample training function which loops over the training set and evaluates the network
    on the validation set after each epoch. Evaluates the network on the training set
    whenever the
    :param images: input images
    :param labels: target labels
    :param fold: tuple of (train, test) index numbers
```

```
:param model_type: model type ('cnn', '1dconv', 'maxpool', 'lstm', 'mix')
:param batch_size: batch size for training
:param num_epochs: number of epochs of dataset to go over for training
:return: none
"""
num_classes = len(np.unique(labels))
(X_train, y_train), (X_val, y_val), (X_test, y_test) = reformatInput(images, labels, fold)
X_train = X_train.astype("float32", casting='unsafe')
X_val = X_val.astype("float32", casting='unsafe')
X_test = X_test.astype("float32", casting='unsafe')
# Prepare Theano variables for inputs and targets
input_var = T.TensorType('floatX', ((False,) * 5))()
target_var = T.ivector('targets')
# Create neural network model (depending on first command line parameter)
print("Building model and compiling functions...")
# Building the appropriate model
if model_type == '1dconv':
    network = build_convpool_conv1d(input_var, num_classes)
elif model_type == 'maxpool':
    network = build_convpool_max(input_var, num_classes)
elif model_type == 'lstm':
    network = build_convpool_lstm(input_var, num_classes, 100)
elif model_type == 'mix':
    network = build_convpool_mix(input_var, num_classes, 100)
elif model_type == 'cnn':
    input_var = T.tensor4('inputs')
    network, _ = build_cnn(input_var)
    network = DenseLayer(lasagne.layers.dropout(network, p=.5),
                         num_units=256,
                         nonlinearity=lasagne.nonlinearities.rectify)
    network = DenseLayer(lasagne.layers.dropout(network, p=.5),
                         num_units=num_classes,
                         nonlinearity=lasagne.nonlinearities.softmax)
else:
    raise ValueError("Model not supported ['1dconv', 'maxpool', 'lstm', 'mix', 'cnn']")
# Create a loss expression for training, i.e., a scalar objective we want
# to minimize (for our multi-class problem, it is the cross-entropy loss):
```

```
prediction = lasagne.layers.get_output(network)
loss = lasagne.objectives.categorical_crossentropy(prediction, target_var)
loss = loss.mean()
params = lasagne.layers.get_all_params(network, trainable=True)
updates = lasagne.updates.adam(loss, params, learning_rate=0.001)
# Create a loss expression for validation/testing. The crucial difference
# here is that we do a deterministic forward pass through the network,
# disabling dropout layers.
test_prediction = lasagne.layers.get_output(network, deterministic=True)
test_loss = lasagne.objectives.categorical_crossentropy(test_prediction,
                                                        target_var)
test_loss = test_loss.mean()
# As a bonus, also create an expression for the classification accuracy:
test_acc = T.mean(T.eq(T.argmax(test_prediction, axis=1), target_var),
                  dtype=theano.config.floatX)
# Compile a function performing a training step on a mini-batch (by giving
# the updates dictionary) and returning the corresponding training loss:
train_fn = theano.function([input_var, target_var], loss, updates=updates)
# Compile a second function computing the validation loss and accuracy:
val_fn = theano.function([input_var, target_var], [test_loss, test_acc])
# Finally, launch the training loop.
print("Starting training...")
best_validation_accu = 0
# We iterate over epochs:
for epoch in range(num_epochs):
    # In each epoch, we do a full pass over the training data:
    train_err = 0
    train_batches = 0
    start_time = time.time()
    for batch in iterate_minibatches(X_train, y_train, batch_size, shuffle=False):
        inputs, targets = batch
        train_err += train_fn(inputs, targets)
        train_batches += 1
    # And a full pass over the validation data:
    val_err = 0
    val_acc = 0
    val_batches = 0
```

```
        for batch in iterate_minibatches(X_val, y_val, batch_size, shuffle=False):
            inputs, targets = batch
            err, acc = val_fn(inputs, targets)
            val_err += err
            val_acc += acc
            val_batches += 1
        av_train_err = train_err / train_batches
        av_val_err = val_err / val_batches
        av_val_acc = val_acc / val_batches
        # Then we print the results for this epoch:
        print("Epoch {} of {} took {:.3f}s".format(
            epoch + 1, num_epochs, time.time() − start_time))
        print(" training loss:\t\t{:.6f}".format(av_train_err))
        print(" validation loss:\t\t{:.6f}".format(av_val_err))
        print(" validation accuracy:\t\t{:.2f} %".format(av_val_acc ∗ 100))
        if av_val_acc > best_validation_accu:
            best_validation_accu = av_val_acc
            # After training, we compute and print the test error:
            test_err = 0
            test_acc = 0
            test_batches = 0
            for batch in iterate_minibatches(X_test, y_test, batch_size, shuffle=False):
                inputs, targets = batch
                err, acc = val_fn(inputs, targets)
                test_err += err
                test_acc += acc
                test_batches += 1
            av_test_err = test_err / test_batches
            av_test_acc = test_acc / test_batches
            print("Final results:")
            print(" test loss:\t\t\t{:.6f}".format(av_test_err))
            print(" test accuracy:\t\t{:.2f} %".format(av_test_acc ∗ 100))
            # Dump the network weights to a file like this:
            np.savez('weights_lasg_{0}'.format(model_type), ∗lasagne.layers.get_all_param_values(network))
    print('−'∗50)
    print("Best validation accuracy:\t\t{:.2f} %".format(best_validation_accu ∗ 100))
    print("Best test accuracy:\t\t{:.2f} %".format(av_test_acc ∗ 100))
```

```
if __name__ == '__main__':
    from utils import reformatInput

    # Load electrode locations
    print('Loading data...')
    locs = scipy.io.loadmat('../Sample data/Neuroscan_locs_orig.mat')
    locs_3d = locs['A']
    locs_2d = []
    # Convert to 2D
    for e in locs_3d:
        locs_2d.append(azim_proj(e))

    feats = scipy.io.loadmat('../Sample data/FeatureMat_timeWin.mat')['features']
    subj_nums = np.squeeze(scipy.io.loadmat('../Sample data/trials_subNums.mat')['subjectNum'])
    # Leave-Subject-Out cross validation
    fold_pairs = []
    for i in np.unique(subj_nums):
        ts = subj_nums == i
        tr = np.squeeze(np.nonzero(np.bitwise_not(ts)))
        ts = np.squeeze(np.nonzero(ts))
        np.random.shuffle(tr) # Shuffle indices
        np.random.shuffle(ts)
        fold_pairs.append((tr, ts))

    # CNN Mode
    print('Generating images...')
    # Find the average response over time windows
    av_feats = reduce(lambda x, y: x+y, [feats[:, i*192:(i+1)*192] for i in range(feats.shape[1] / 192)])
    av_feats = av_feats / (feats.shape[1] / 192)
    images = gen_images(np.array(locs_2d),
                                  av_feats,
                                  32, normalize=False)
    print('\n')

    # Class labels should start from 0
```

```
print('Training the CNN Model...')
train(images, np.squeeze(feats[:, -1]) - 1, fold_pairs[2], 'cnn')

# Conv-LSTM Mode
print('Generating images for all time windows...')
images_timewin = np.array([gen_images(np.array(locs_2d),
                                                feats[:, i * 192:(i + 1) * 192], 32, normalize=False)
                                                    for i in
                                      range(feats.shape[1] / 192)
                                      ])
print('\n')
print('Training the LSTM-CONV Model...')
train(images_timewin, np.squeeze(feats[:, -1]) - 1, fold_pairs[2], 'mix')

print('Done!')
```