THE COOPER UNION FOR THE ADVANCEMENT OF SCIENCE AND ART
ALBERT NERKEN SCHOOL OF ENGINEERING

Automatic Artifact Annotator for EEG Waves Using Recurrent and Convolutional Neural Networks

By

DongKyu Kim

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

April 28, 2019

Advisor

Sam Keene

THE COOPER UNION FOR THE ADVANCEMENT OF SCIENCE AND ART
ALBERT NERKEN SCHOOL OF ENGINEERING

This thesis was prepared under the direction of the Candidate's Thesis Advisor and has received approval. It was submitted to the Dean of the School of Engineering and the full Faculty, and was approved as partial fulfillment of the requirements for the degree of Master of Engineering.

_____

Barry Shoop – Date

Dean, School of Engineering

_____

Prof. Sam Keene – Date
Candidate's Thesis Advisor

Acknowledgment

I would like to thank my advisor, Professor Sam Keene, for his support, encouragement, and guidance that he has given me to go through the thesis process. Without him, I would never have learned and developed an interest in machine learning.

I would like to thank Professor Chris Curro, who taught me deep learning, and Professor Ian Kremenic, who taught me the basics of medical signals, especially background on the electroencephalogram.

I would also like to thank the electrical engineering faculty members at The Cooper Union for giving me a good education and always challenging me. I would like to give special thanks to Professor Fred Fontaine, who approved all of my overloaded schedules for the past 3 years.

I would like to thank my family members who have provided me mental supports when I needed them.

I would like to thank my friends, and classmates, especially Junbum Kim, and Samuel Cheng, for supporting me throughout my four years at the Cooper Union.

Abstract

Electroencephalogram (EEG) is one of the widely used non-invasive brain signal acquisition techniques that measure voltage fluctuations from neuron activities of the brain. EEG is typically used to diagnose and monitor disorders such as epilepsy, sleep disorders, and brain death and also to help advancement of various fields of science such as cognitive science, and psychology. Unfortunately, EEG signals usually suffer from a variety of artifacts like eye movements, chewing, muscle movements, and electrode pops, which disrupts the diagnosis and hinders precise representation of brain activities. This thesis evaluates three deep learning methods, and an ensemble method to detect the presence of the artifacts and to classify the kind of the artifact to help clinicians resolve problems regarding artifacts immediately during the signal collection process. Models were optimized to map the 1-second segments of raw EEG signals to detect 4 different kinds of artifacts. Among all the models, the best model is the ensemble model, which achieved 5-class classification accuracy of 67.59%, and a true positive rate of 80% with 25.82% false alarm for binary artifact classification with time-l. The model is lightweight and can be easily deployed in portable machines.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

The study of the brain, neuroscience, to understand about ourselves better has been a great research area that combines scientists and engineers across various disciplines. Sometimes, the understanding of the basis of learning, perception, and consciousness is described as the "ultimate challenge" of biological sciences [1]. A lot of advancements in neuroscience come from analyzing recordings of the brain. However, due to the overwhelming amount of electrochemical activities in brains, the collection of reliable data is still one of the biggest challenges in neuroscience [2].

There are two main branches of brain signal acquisition methods: invasive methods, and non-invasive methods. Invasive methods involve placements of electrodes inside the brain, or insertion of needles through the subject's head to collect precise and highly local data. On the other hand, non-invasive methods such as electroencephalogram (EEG), and magnetic resonance imaging (MRI) suffer from noise and various artifacts [2]. Due to high interests and potentials in this area of research, and relatively cheap and easy access to EEG machines [3], there is a large amount of data available for analysis. However, a lot of EEG data suffer from artifacts that are both physiological and technical, and the artifacts are usually not documented well [2]. If there is a model that can distinguish between artifacts, and cerebral data automatically, neuroscience can advance

further as reducing the effect of artifacts will increase the signal to noise ratio so that brain activity can be detected more precisely.

To achieve this goal, Temple University has constructed a large dataset of EEG waves from various patients, specifically labeled for artifacts [4] to help engineers and scientists to build models that combat the lingering artifacts. Previously, Golmohammadi, and colleagues [5] developed a model that automatically analyzes EEG signals to help physicians diagnose brain-related illnesses such as seizures using hybrid deep learning architectures. This model integrates hidden Markov models (HMMS), deep learning models, and statistical language models to deliver a composite model that has a true positive rate of 90% while having a false alarm rate of below 5% on events of clinical interests: spike and sharp waves, periodic lateralized epileptiform discharges, and generalized periodic epileptiform discharges [5]. This model proves the viability of big data and deep learning methods in detecting events in EEG signals. More works in this area are documented in the next chapter.

The work in [5] attempts to classify artifacts as well as the mentioned events of clinical interest, but the model developed was only able to distinguish 14.04% of the artifacts correctly from the data. As the goal of that model was to detect seizures and epilepsy, no further analysis on artifacts was done, but it was noted that transient pulse-like artifacts such as eye movements, and muscle

movements can significantly degrade the performance [5]. In this thesis, a method that can quickly identify the presence of artifacts and the type of the artifacts during the collection process is proposed, so that a clinician can resolve the problem immediately and ensure the collected data is artifact-free. In order to achieve this goal, multiple deep learning models with varying model size, inference time, and accuracies were developed and optimized to compare and contrast between advantages and disadvantages of different approaches. The key feature of the models is all the inferences are done directly on the raw signals such that there is no need for preprocessing the data other than aggregating enough samples to be used for predictions by the model. The system aims to be memory efficient, and computationally light, while being fast enough to be implemented on portable systems such as Raspberry Pi and detect and classify artifacts in real-time, potentially in a clinical setting.

The rest of this thesis is organized as follows. In Chapter 2, previous works on automatic artifact detectors for EEG signals are presented. In Chapter 3, background information on machine learning, deep learning, and the human brain as well as electroencephalogram (EEG) is presented. In Chapter 4, details of the dataset that was used for the experiment, descriptions of how data were modified and shaped, and motivations regarding certain choices are presented and discussed as well as the experiment setup. In Chapter 5, the results of the experiments are presented and analyzed. Finally, in Chapter 6, the conclusion

regarding the experiment is presented to summarize the findings and suggest future works. In Chapter 7, a list of references is presented, and in Chapter A, appendix, code samples to support the thesis are presented.

2. Previous Works

There have been numerous efforts to combat the artifact problems in EEG signals. A lot of research has been done to reduce the effects of artifacts by utilizing prior knowledge such as how some artifacts behave in the signal. Artifact removal and detection tools of this nature tend to examine the statistical characteristics of the signals.

Nolan, Whelan, and Reilly [6] proposed FASTER, Fully Automated Statistical Thresholding for EEG artifact Rejection, which uses independent component analysis (ICA) to separate EEG signals into neural activity and artifacts. ICA works by separating multivariate signals into additive subcomponents by assuming different subcomponents are statistically independent of each other. The advantage of using ICA is that it tries to reduce the statistical dependencies of different components of the signal, trying to orthogonalize the components [7]. After decomposition, the model uses a series of statistical comparison charts to check for statistical features such as correlation with signal components separated using ICA, mean, variance, spatial, and et cetera. This model was tested on simulated EEG and real EEG and had a true positive rate of over 90%. The main drawback of this model is the computational time as the ICA decomposition takes about 40 minutes. Nevertheless, the model not only detects the signal quite accurately but also can remove the signal, as the

artifact component of the signal can be extracted. This model can detect eye movement, EMG artifacts, linear trends, and white noise.

Similarly, Singh and Wagatsuma [8] used Morphological Component Analysis (MCA), which uses a dictionary of multiple bases to guarantee reconstruction of signals. MCA is applied to the EEG signal to deconstruct into a combination of bases in the dictionary. Singh and Wagatsuma hypothesized that three dictionaries of bases are dominant, and they are undecimated wavelet transform (UDWT), discrete sine transform (DST), and DIRAC (standard unit vector basis) [8]. The decomposition was able to show that EEG signals and their artifacts are represented by different dictionaries of bases, indicating that given the decomposition result, these can be distinguished. They successfully categorized which dictionary corresponded well with the signal or the artifact. This research demonstrates that an ensemble of different signal processing could work for artifact classification. There are numerous other additional statistical approaches to separate the real EEG signal from the artifacts, such as canonical correlation analysis, which Clercq used to remove muscle artifacts from the EEG signals [9].

All of the statistical approaches of the problem requires a deconstruction of EEG signals into multiple components and analyzing each component to determine which components are responsible for artifacts and which are

responsible for the real signal. Though they are highly interpretive, the separation procedure takes a lot of computations, and prior knowledge, such as the number of artifacts, a set of orthogonal bases that work well with time-series data or general behavior of artifacts is required. Due to the complex nature of the EEG signals, deep learning with its ability to learn hidden features from the raw data has shown great promises [10].

According to the review paper by Roy et al [11], among the 156 papers that the authors reviewed from January 2010 to July 2018 about applying deep learning to EEG signals, some applied data preprocessing techniques and artifact rejection techniques such as ICA mentioned above to combat the artifacts, while some just used the raw data. Given that the majority of the papers did not use any artifact removal schemes, Roy et al suggest that using deep learning on EEG data might avoid the artifact removal step without performance degradation. However, all the papers mentioned in Roy's review paper specifically target certain applications such as epilepsy, sleep monitorings and brain-computer interfaces. None of the papers mentioned targets the detection of artifacts specifically. However, the paper guides this research in a certain way as 40% of the studies use convolutional neural networks (CNNs), and 14% use recurrent neural networks (RNNs) [11].

Other works relating to deep learning and EEG signals or EEG like signals not mentioned in the review paper above include Krishnaveni et al's work on ocular artifacts removal in EEG signals [12] and Hasasneh et al.'s work on automatic classification of ocular and cardiac artifacts in magnetoencephalography (MEG) [13]. Both of them include some data preprocessing like ICA for Hasasneh's work, and the Joint Approximation Diagonalisation of Eigen matrices (JADE) algorithms for Krishnaveni's work to separate the real signal from the artifact signal before using neural networks. The detection rates for test data for these works are 94.4%, and 92% respectively for Hasasneh's work [13] and Krishnaveni's work [12]. However, both of these works only address a single or two kinds of artifacts at the same time, while the model to be proposed will include 4 different artifacts to be classified separately with no preprocessing such that the model can be applied directly to the raw data.

There have been many attempts and successful attempts in detecting artifacts and classifying them using statistical machine learning and inferences, but there are not many pieces of research being done doing the same using deep learning. Deep learning approaches are particularly adept at optimizing an arbitrary large model and recognizing complex patterns [10]. The previous methods require mathematical models for artifact events or seizure events to classify the signals accurately, hence the performance of the models depends highly on the accuracy of the proposed mathematical models. However, usage of

deep learning models can alleviate the incorrect modeling error as no accurate model is needed to classify different events. In addition, statistical analysis of large temporal data is computationally heavy and takes a long time. While training a deep learning model to optimize the parameters may take a long time, inference time for the completed models is relatively small compared to statistical methods. To use these advantages, many works have attempted to classify different aspects of the EEG signals for monitoring purposes for seizure or sleep using deep learning, but not a lot of work has been done in detecting and classifying artifacts using deep learning, especially classifying multiple artifacts instead of detecting a small number of artifacts.

3. Background Information

3.1 Machine Learning

Machine learning (ML) is the study of algorithms and models that utilize computer systems to perform pattern recognition tasks without using explicitly writing down the true pattern behind the tasks. Pattern recognition tasks are problems of searching for hidden rules or regularities of data automatically using computer algorithms. Due to fundamentally similar natures, Bishop says in his book [14], "these activities can be viewed as two facets of the same field". Machine learning approach usually consists of using large amounts of data to improve the model's performance in certain tasks. An ideal machine learning model generalizes the problem based on the available data such that the model will be able to predict accurately on unseen examples. As these models, given enough data, can perform really well in certain tasks such as a variety of image classification tasks, including handwriting recognition, and house number recognition [15], or playing games [16]. There are many types of machine learning algorithms as there are many tasks to be tackled, and not all the tasks can be solved using a single method. In the following subsection, supervised learning, which is the nature of the classifier that was built for this thesis.

3.1.1 Supervised Learning

Supervised learning is a machine learning system that builds a model that learns the input-output relationship of a system of interest in which both input features and desired output labels are provided. The input and output data are prelabelled for classification and are available for the model at training so that this data can be used as the learning basis for the future unseen data.

Mathematically supervised learning algorithms work as follows. Given a set of training data pairs $\{(\vec{x}_i, y_i) \mid i = 1,2 \ldots, N\}$, where $\vec{x}_i$ is the input feature vector of arbitrary but finite length, and y_i is its corresponding label, the learning algorithms try to find a function or a system that maps $X \rightarrow Y$, where X is the input space, and Y is the output space. Let the system that maps the input to the output to be a function g, such that $\widehat{y_i} = g(\overrightarrow{x_i})$, represent the estimator of y_i. The machine learning algorithms learn this function through minimizing a loss function L that usually computes how close the estimated labels, $\hat{y}$ are to the true labels, y.

All of the machine learning models that are going to be discussed in this thesis utilizes the loss function to optimize the machine learning algorithms to best represent the unknown system. All the algorithms use gradient-based methods to optimize the parameters associated with the models.

3.2 Deep Learning

Deep learning is a powerful framework in machine learning with the ability to represent systems of extreme complexity [10] due to its own model complexity from multiple artificial neural network layers which will be discussed in the later sections.

3.2.1 Artificial Neural Networks

Artificial neural networks, ANNs, are computing networks that are loosely inspired by the neural networks of animal brains. The idea behind the neural networks of animal brains is that multiple neurons are interconnected by synapses to perform a task together. Neurons are interconnected such that a piece of information propagates through each other to deliver certain information to certain parts of the body to perform each task. More details will be presented in the neuron section of this chapter. Inspired by the animal neurons, artificial neural networks have nodes that are interconnected with each other as in Figure 1.

The input layer usually is reserved for the external system that presents the network with a feature vector. Depending on the length of the feature vector, the number of input nodes, or neurons, is adjusted. All the input nodes represent

an element in the feature vector, and in networks with fully connected hidden layers, each of the input nodes is connected to every node in the hidden layer

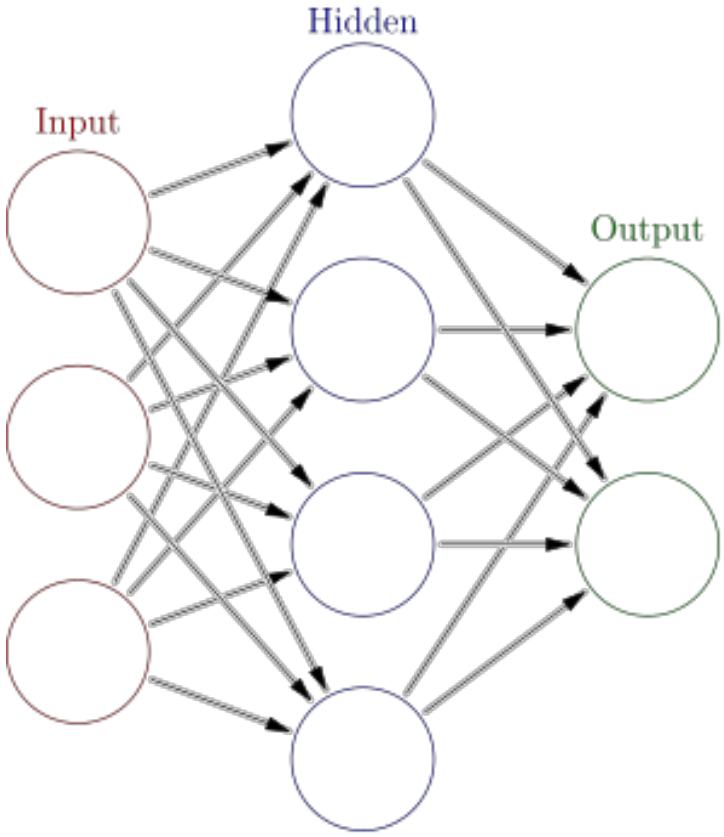

Figure 1: Representation of an Artificial Neural Network [17]

just as in Figure 1. Although the figure only has one hidden layer, the number of the hidden layers can be arbitrarily large, and this number is usually referred to as the depth of the neural network. Again, in a fully connected layer, each node in a hidden layer is connected to every node in the next hidden layer. The output layer occurs at the end of the neural network, and each node of the output layer

is connected to every node in the hidden layer from the last step. The nodes in the output layer represent the label or the target vector, and for this thesis, the output layer will represent the confidence level of different artifacts, indicating how sure the model think each label exists in the give EEG signal.

The way each node is connected is through linear combination with an activation function. Each path from a node to another node has a certain weight and bias. Using the input layer and the hidden layer in Figure 1 as an example, each element in the input layer is denoted as $\{x_1, x_2, x_3, \dots, x_n\}$, and each element in the hidden layer which will be regarded as the output elements is denoted as $\{y_1, y_2, y_3, \dots, y_m\}$. Then each output element is computed by the following equation:

$$y_i = f\left(\sum_{ii=1}^{n} (x_{ii} * h_{ii} - b_{ii})\right) \; for \; i \; in \; 1,2, \dots m$$

h_{ii} and b_{ii} represent the weight and the bias associated with the input node ii, and the function f represents an arbitrary activation function. The role of the activation function is to introduce non-linearity to the network as without the non-linearity, all the linear combinations connecting the input nodes to the output nodes, regardless of how many hidden layers there are, can be summarized into a single multiplicative matrix, and a corresponding bias. In other words, without the activation function, adding many hidden layers does not change the network.

There is a large number of activation functions that are capable of introducing non-linearity to the network. Nwankpa et al. have done thorough research on different kinds of activation functions [18]. Traditionally the sigmoid activation was used which is given by the following equation.

$$f(x) = \frac{1}{1 + e^{-x}}$$

This equation is named "sigmoid" as when the equation is plotted, the graph is S-shaped. The main advantage of this activation function is that since it has a positive derivative everywhere with smoothness, it is a respectable measure for probability based outputs, as it maps the whole real axis into a finite interval between 0, and 1 [14]. This function is usually used in the outermost layer of the binary classification task to convert the logits into confidence levels. However, since the softmax function which will be introduced later, is a generalization of the sigmoid function, in this thesis sigmoid function is not used. The main drawbacks of this activation function are sharp gradients during backpropagation and slow convergence.

Another activation function is the rectified linear unit (ReLU) activation function. The ReLU function is defined as follows:

$$f(x) = \max(0, x)$$

The function "rectifies" values below 0 by forcing them to be 0. The main advantage of this activation function is that it is computationally light. When the input value is positive, the function just returns the value, and if the input value is negative, the function returns 0. In addition, its gradient is either 0 or 1, which is also easy to compute. The major drawback of this activation function is that since the gradient of the negative values is all 0, with some bad weight initializations, the model can never learn, and be stuck at the same state. There are additional derivates of this activation function such as the Leaky ReLU (LReLU), Parametric Rectified Linear Unit (PReLU), or Randomized Leaky ReLU (RReLU) [18]. However, for lighter computation, for this research, only ReLU was used.

The last activation function that will be introduced is the softmax function. This activation function is used to make sense of the outer most layers of models of classification tasks. All the models that are developed in this thesis have a sigmoid activation function for the last fully connected layer to convert all the output logits as confidence levels that are comparable. The softmax function is computed using the following equation:

$$f(x_i) = \frac{e^{x_i}}{\sum_{ii} e^{x_{ii}}}$$

The characteristic of this function is that it converts a vector of real numbers into probabilities. This function is used in the outermost layer of all the models in this

thesis, to convert the logits into probabilities or confidence levels so that the classification task can be done.

With these activation functions, the fully connected layers in artificial neural networks can introduce non-linearities in the model to learn arbitrary complex mappings between the given inputs and outputs. Artificial neural networks learn by setting up a loss function or an evaluation function that examines the predicted output and backpropagate the gradient of the loss function to the input layer. Backpropagation updates all the weights in the neural networks such that in the next epoch, the model can predict better.

There is one additional layer that is used before fully connected layers in this thesis, the flatten layer. This layer reshapes the input feature vector by reducing the number of dimensions to one, regardless of the original number of dimensions. For example, if the input vector has a shape of (32,16,16,3), which means the vector has 4 dimensions with 32 elements in the first dimension, 16, 16, 3 in the second, the third, and the fourth dimensions respectively. The role of the flatten layer is to convert this input vector into one long vector of shape (24576,). This new vector contains all the information that was available in the original input vector but only has one dimension. This layer removes dimensionality in data frames to be used for fully connected layers.

3.2.2 Convolutional Neural Network

Convolutional Neural Network (CNN) is a derivation of the standard ANN that is optimized for problems with localities. The motivation for this structure is from Hubel and Wiesel's work on cat's visual cortexes [19]. As the visual cortexes contain two basic types of cells that respond to certain shapes and are decent image processing system, a similar structure was implemented to create a convolutional layer. Convolutional Neural Networks consist of multiple of convolutional layers that work as shown in Figure 2.

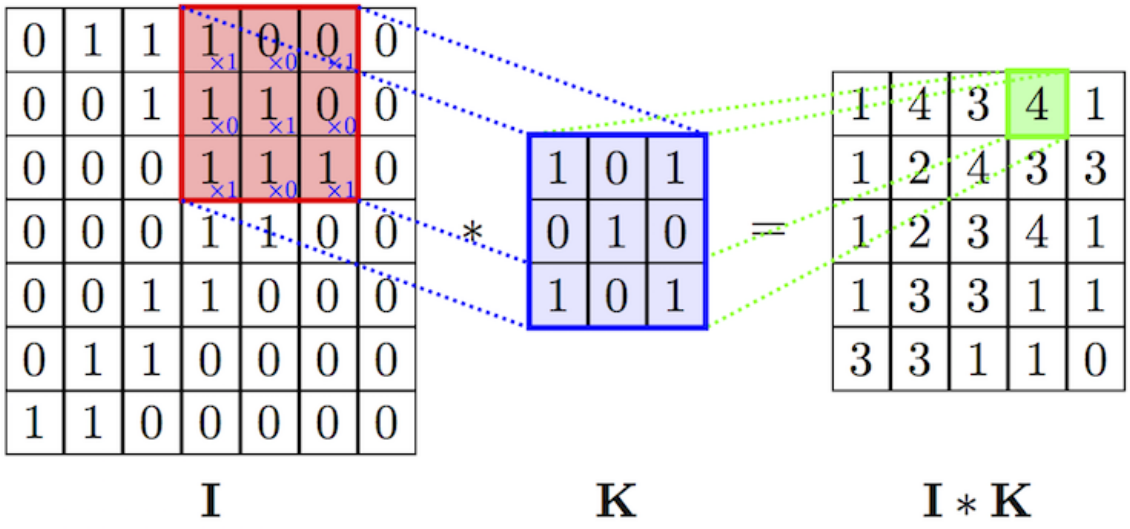

Figure 2: Visualization of Convolutional Layer [20]

The convolutional layer consists of a collection of filters that convolves around the input image. Using the figure as an example, K is one of the filters defined in the layer, and it moves around the input image, I, such that the K never leaves the boundary of I, but covers the whole area of I. The filter acts as a mask, which does element-wise multiplication with a region selected in the input image. After the element-wise multiplication, all the resulting values in the box are added to produce an output value. There is a user definable variable named stride, which defines how long the filter travels before the next observation. For the case of the example, the stride is 1 because the filter shifts 1 unit in each direction. As there are many filters available in a single layer, the layer can "look" at the input image in many different ways, extracting useful features using different views. Hence, the goal of this layer is to view images or sequences of data better, so the coefficients of filters adapt to the task that the layer has been presented to.

However, the above example is on images, in which 2 dimensions are available for the input. The signals of interest of this research are time-series signals with multiple channels. Hence, only 1-dimensional convolutional layers are used. The only difference is the filters are 1-dimensional instead of 2-dimensional, and the filter slides through different channels but not across the temporal axis.

Additional deep learning layer of interest is max pooling layer. A visual representation of a Max-pooling layer is shown in Figure 3. The example shown is a 2 by 2 max-pooling layer. Max-pooling layer basically downsamples an image or an array by a specified factor. If an image is max-pooled with a stride of 2 as is the case in the figure, the image is divided into tiles of 2 by 2 blocks, and only the maximum value of the block is retained and saved to form a sampled version of the original image. Again the example given is 2-dimensional for visual aid, but the signals of interest are one dimensional in channels. Hence, for a 1-dimensional max-pooling layer, instead of reducing the size in two dimensions, only 1-dimension will be reduced.

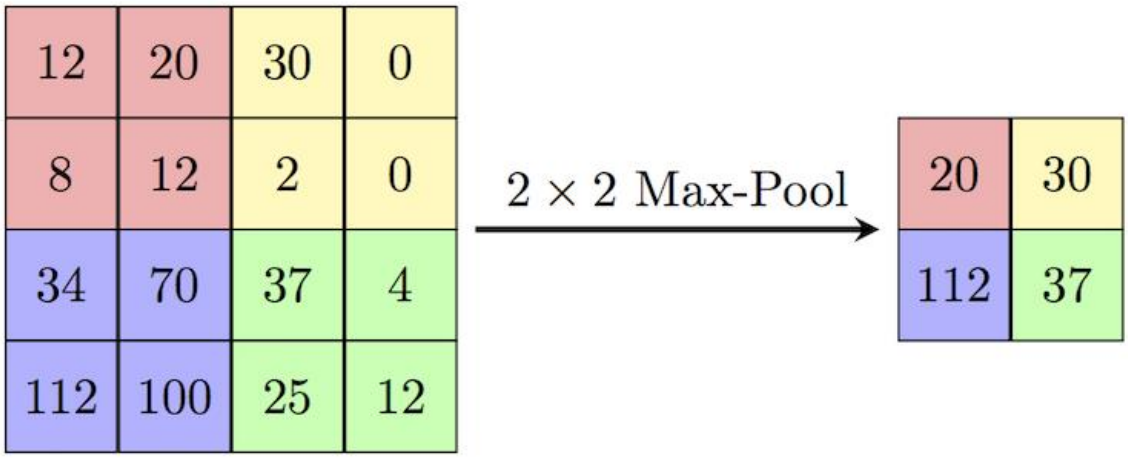

Figure 3: Representation of Max-Pooling Layer [21]

The purpose of this layer is to retain important features while decreasing the size of the representation. For the purpose of the thesis, max-pooling layers are used as literal down-sampling systems as they were only used to reduce the temporal dimension.

3.2.3. Recurrent Neural Network

Another form of neural network architecture that will be used is Recurrent Neural Network (RNN). Recurrent Neural Networks are like standard Artificial Neural Networks except, RNNs have access to the previous outputs as well as the current input. The existence of feedback loops makes RNNs excellent at processing temporal data such as speech recognition [22] or electroencephalogram (EEG) or functional MRI. One of the major drawbacks of this class of network is that since input includes the output from the previous iteration, the rope back to the past goes all the way back to the beginning of the sequence. This gives loss to have access to all the weights that were ever engaged with the network, such that when the backpropagation begins, the loss gets multiplied over and over by the countless weights from the past. This causes either the loss to be 0 when most of the weights' magnitudes are below 1, or causes the loss gradient to be really large. This problem is common difficulty found in training deep learning models, and usually called "the vanishing gradient problem", or "the exploding gradient

problem" depending on whether the gradient goes to 0 or some large number [23]. As a result, the model either does not learn anything or fluctuates too much to settle down and learn anything.

In order to combat the vanishing or exploding gradient problems [23], a Recurrent Neural Network architecture called Long short-term memory (LSTM) was proposed by Hochreiter and Schmidhuber in 1997 [24]. This architecture works as a memory cell that has an input gate, an output gate, and a forget gate. All of the gates have access to the current input and the previous output value, but each gate has its own criteria for choosing whether the information will pass through the gate or not. The gates modulate interactions between the memory cell and the environment. In essence, the LSTM layer keeps a cell that contains certain information, with three controllers named input, output, and forget gates change the information contained with respect to the outside input. This enables the LSTM layer cells to remember information for an arbitrary duration without chaining all the time steps between the present and the past.

3.3 The Human Brain

The brain is the organ located in the head that controls the body by sending electrochemical signals from neurons. It is responsible for most of the activities in the body, as well as processing the information human receives from sensory

organs. The functions of the brain mainly include motor control, sensory information processing, regulation of body conditions, as well as emotional controls. The brain is capable of doing all the tasks due to a large number of neurons and their interconnectivity.

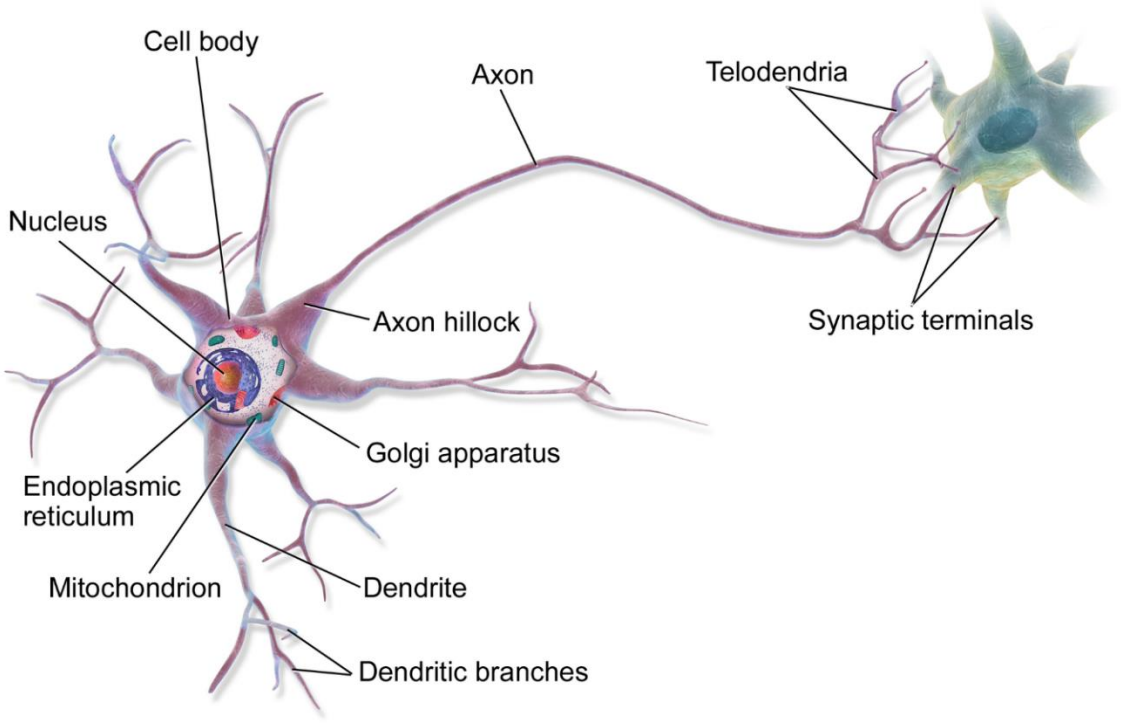

Figure 4: Diagram of a Neuron [25]

3.3.1 Neuron

A neuron is a cell that communicates with other cells using synapses, which are special connections between cells. Usually, a neuron receives information from

other neurons through dendrite and transfers the information using axons. In order to transfer information, neurons have to generate an action potential which rapidly shoots along the axon, and activates synapses on the other side when the action potential reaches them.

3.3.2 Electroencephalogram (EEG)

An electroencephalogram (EEG) is a test done to usually diagnose or monitor problems in the brain by measuring electrical activities of the brain created by action potentials. First of all, EEG is a non-invasive technique that records the electrical activity by positioning electrodes over the scalp of the subject [26]. The number of electrodes can vary, but in this thesis, the configuration of interest is the international 10-20 system, which is portrayed in Figure 5. The international 10-20 system's name means that the electrodes are placed on locations dividing perimeters into 10% and 20% intervals.

All the electrodes are referenced to nasion (front), inion (rear), and the vertex, which is where CZ is from the diagram. The 10%, and 20% are relative distances to different lengths depending on the electrode. The distance between A1 to T3 and A2 to T4 are each 10% of the length of the line connected A1, and A2 and all the distances between electrodes on the line are 20% of the same length. The electrodes around the head, (FP1, FP2, F8, T4, T6, O2, O1, T5, T3, F7 on the

diagram) are placed such that the distance between neighboring electrodes is 10% of the circumference of the head. Other electrodes that are not on the line, or on the circle are placed appropriately such that the distance between neighboring electrodes is approximately the same. 21 electrodes are used to cover all the areas of the head in this configuration. There are other variations of this configuration for better resolution of recordings such as a configuration that uses 74 electrodes to cover every 10% location and a configuration that uses over 300 electrodes to cover every 5% location [27].

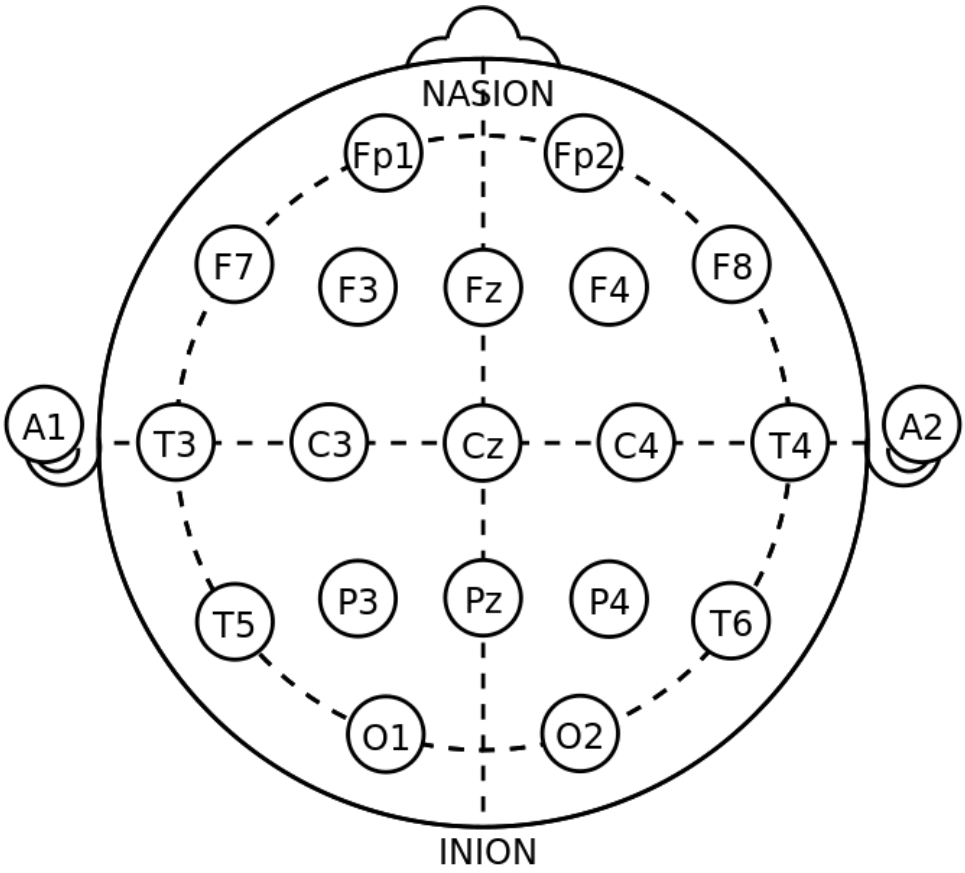

Figure 5: Electrodes of the International 10-20 System for EEG recording [28]

For the measurements that created the dataset of interest, referential montage was used, which means two electrodes were used to compute the electrical activity of a region in the brain. Corresponding montages and electrodes are listed in Table 1.

EEG signals are very low amplitude signals such that 1-10 μV can be considered a decent signal, while it can get up to several hundred μV [29]. The signals usually consist of 4 different waves: delta, theta, alpha, and beta waves. Delta wave has a frequency of 3Hz or below, theta wave has a frequency of 3.5-7.5 Hz, the alpha wave has a frequency of 7.5-13 Hz, and the beta wave has a frequency of more than 14 Hz [30]. Each of the waves has different amplitude characteristics and physiological meanings. For example, theta waves are perfectly normal to be present in children up to age 13, and adults while in sleep, but the presence in awake adults indicates health problems [30]. Unfortunately, EEG signals suffer from numerous artifacts, which can be patient-related such as movement, sweating, eye movements, but can also be non-patient related such as 50/60Hz wall outlet artifact, cable movements, and electrodes popping out [2]. The goal of this thesis is to automatically detect these artifacts.

4. Experiment

4.1 Resources

The dataset used in this thesis is the Temple University Hospital's EEG Artifact Corpus. The dataset was developed to help to reduce the harmful effects of artifacts for EEG event classification algorithms such as seizure detection algorithms. The version of the dataset is v1.0.0, and the dataset is derived from the v1.1.0 of the TUH EEG Corpus [4]. There are 310 observations with 213 patients with varying durations and sampling rates. The methods used to combat variations in the dataset are detailed in the upcoming sections.

The version of the python that is used is 3.6.8. Additional libraries used are matplotlib v3.0.2, numpy v1.16.0, tqdm v4.31.1, pandas v0.24.0, pyedflib v0.1.14, scipy v1.2.0, tensorflow v1.12.0, and keras v2.2.4. The experiments were done using a machine equipped with 16GB memory, AMD FX(tm)-6300 Six-Core Processor 3.5GHz, and a GeForce GTX 1070 8GB graphics card. The data drive in which the corpus was located in was a standard hard drive with 7200RPM. Finally, the environment was a Windows 10 environment with a virtual environment with all the above libraries created using conda for the Anaconda Python distribution.

4.2 Data Preprocessing

The dataset contains 3 different configurations of EEG: AR (averaged reference), LE (linked ears reference), and AR_A configuration that is a modified version of the AR configuration. All the data contain standard measurements that one could expect from the 10-20 International System. For AR, and LE configurations, 22 montages can be derived from the available channel information, and for AR_A configuration only 20 montages can be derived as EEG A1-REF and EEG A2-REF channels are missing. The computation necessary to derive the montages is tabulated in Table 1.

There are only 7 occurrences of AR_A configuration with 4 patients, and as this configuration lacks similarity to other configurations, it was discarded for the experiments. Hence, there are 303 observations with 209 patients available. There are techniques to fill in the missing channels using adjacent channels as described in [6]. However, since missing A1, and A2 electrodes in AR_A configuration are located on the outermost boundary of the head, limiting the accuracy of any interpolated guesses, hence AR_A configuration observations were discarded.

Montage Number	Computation
1	FP1-F7
2	F7-T3
3	T3-T5
4	T5-01
5	FP2-F8
6	F8-T4
7	T4-T6
8	T6-02
9	A1-T3
10	T3-C3
11	C3-CZ
12	CZ-C4
13	C4-T4
14	T4-A2
15	FP1-F3
16	F3-C3
17	C3-P3
18	P3-01
19	FP2-F4
20	F4-C4
21	C4-P4
22	P4-02

Table 1: List of Montages with Appropriate Computation

The original data are in The European Data Format (EDF), which is a standard file format designed for storage of medical time series data. All the EDF files provided have all the electrode information so that montages defined in the instruction that came along can be derived easily. In addition, corresponding label files, which have artifact class labels for the whole EEG session, and for each

montage. There are 7 possible labels in the Temple University's original data corpus: eye movement (eyem), chewing (chew), shivering (shiv), electrode pop, electrode static, and lead artifacts (elpp), muscle artifacts (musc), background noise (bckg), and undefined annotation (null), which is the normal, unaffected signals. The label files provide the start time and stop time in seconds, and the label and the probability which is the confidence level of the label. All the labels in this dataset have a confidence level of 1. The background noise (bckg) label is not available for this dataset, hence this specific artifact dataset has 6 valid labels in total, of which 5 are artifacts and 1 is a normal case.

The EEG sequences had varying sampling frequencies of 250Hz, 256Hz, 480Hz, and 500Hz. As neural network models require input features to be of the same size and having different sampling rates for temporal data can harm the performance of the model, all the signals were resampled to 250Hz, which is the lowest sampling rate. Then the signals were chopped into 1-second segments in order to determine which segment of the signal is affected by artifacts. The 1-second segment was chosen as the lowest frequency of the wave is around 3Hz, which allows the chunk to at least see 3 occurrences of the wave. In addition, all the observations end at a whole second, so that there is no overlap of information or loss of information when the time window for segments is 1 second.

Label	eyem	chew	shiv	elpp	musc	null	total
Occurrences	7471	2727	1338	2663	4892	327222	344975
Percentage (%)	2.17	0.79	0.39	0.77	1.42	94.85	100

Table 2: Number and Percentage of Examples of Each Label

After this process on all 303 observations of varying lengths, a total of 346313 examples were created. The breakdown of the number of examples for each label is given in Table 2. There is a high imbalance of data due to a large number of examples with label null. This is due to the nature of the signal as artifact content in the clinical EEG waves collected is low. There are only 1338 observations of shivering (shiv) label, which consists of 0.39% of all the data available. Due to the relatively low frequency of occurrence, this label only caused problems in developing models as when the data set was split into train, test, and validation set, depending on the random state of the machine, often times shivering label was missing in one of the split data sets. The illustration of this problem is documented in the first section of the result section. Only one model, the RNN based model, was trained and tested on the data without shivering label removed, but the problem is still evident in the example. For the purposes of a more fair evaluation of models, label "shiv" was excluded from the experiments.

Label	eyem	chew	elpp	musc	null	total
Occurrences	7471	2727	2663	4892	327222	346313
Percentage (%)	2.16	0.78	0.77	1.41	94.49	100

Table 3: Number and Percentage of Examples of Each Label After Reduction

The dataset was divided into train set, validation set, and test set. The ratio among the three is 0.75:0.10:0.15. The data set division was done on the unique patient ID, in order to ensure that training and testing were not performed on the same patient as the goal of this model is to generalize to detect artifacts on new patients. Out of 209 patients, 157 patients were allocated to the training set, 21 patients were allocated to the validation set and 31 patients were allocated to the test set. This translates to 224 observations in the train set, 23 observations in the validation set, and 56 observations in the test set. The order of the patient ID has been shuffled before dividing into 3 separate sets.

In addition to the sampling rate, the dynamic range of signals varied. The neural network models prefer data that are around (-1,1) range so all the data were normalized to have 0-mean and standard deviation of 1. In order to retain the relative amplitude scale between each channel, and to account for the fact that statistics of the future data when implemented in a device, would not be available, the mean and the standard deviation of the whole training data was used for the normalization. The mean value of the training data of the sampled version of the dataset that was used for training was -1.5977595, and the standard deviation

was 219.39517. In order to normalize, mean was subtracted from all the data points, and the resulting values were all divided by the standard deviation. Another approach possible would be using the statistic of the 1-second window to normalize across all the channels. However, this approach can potentially lose valuable information such as any information about the absolute magnitude, which could be important in distinguishing between different muscle artifacts, which might cause a great amplitude change or not.

In order to evaluate the models with an environment that is similar to the actual clinical setting, a data-set with the same pre-processing steps was created. The only difference is that this data set contains all the "null" artifact labels. This data set is only used to test the binary classification problem that detects artifacts that will be discussed in the following chapter.

All the edf observations are in 16-bit floating point, however as the tensorflow library does not work on 16-bit precision floating points, all the data after the processing were all converted to 64-bit floating point. Converting all the data to 64-bit floating point and saving them as numpy array objects increased the size of the dataset from 5.39GB to 14.2GB. From experiments, it was evident that extra precision degraded the performance of the training process as the speed of hard drive reading could not keep up with the speed at which the model training

required them. In order to combat this problem, all the data was converted to 32-bit floating point, and this decreased the size of the whole dataset to 7.1GB.

As the interest of the research is to have a fast, on-line automatic annotator for artifacts, no further signal processing techniques or artifact removal techniques currently available were applied. All the data preprocessing steps are done in python, and the appropriate code samples are shown in Appendix A.1.1.

4.3 Models

In this section, final models will be presented as well as all the engineering decisions made on the way. All the failed attempts with explanations and conjectures of why they did not work are presented as well.

4.3.1 Preliminary Studies

In order to examine the data set to learn general characteristics and behavior, a computational graph with 2 fully connected layers was built. The input layer was flattened to reduce the dimension so that the fully connected layer can access all the data. Each fully connected layer had 1024 nodes and was activated by ReLu. Adam [31] optimizer was used, with the default setting, whose learning

rate is 0.001, beta 1 value is 0.9, beta 2 value is 0.999 with no decay. Adam optimizer was used for all the experiments as it is computationally efficient, with little memory requirement [31]. This fully connected model was trained using the training set for 10 epochs with the batch size of 32. The model was validated using the validation set created, but this model was never tested with the test set. The loss function that was used is "categorical_crossentropy", which is defined as below:

$$L_i = -\sum_{n=1}^{N} \left(y_{i,n} log(\hat{y}_{i,n}) \right)$$

i denotes the index of the observation, and n denotes the class label. y and $\hat{y}$ represent the true label and the estimated probability of the label respectively. This is categorical cross-entropy for N number of classes. The model minimizes this loss function by maximizing the estimated probability of class when the true label for the class is one. The model trains completely with an accuracy of 94.4%, which is around the accuracy that one will get with a baseline classifier that guesses all the signals as "null" that yields an accuracy of around 94.5%. The relative frequencies of labels other than "null" were so insignificant, that the model never attempted to optimize the parameters to account for artifact labels. This was evident in the behavior of the test and validation losses and accuracies which just fluctuated a bit over the 10 epochs.

In order to combat the label-imbalance problem, another data set was prepared. In this data set, the "null" labels sub-sampled such that every 30 "null" observation is included to the dataset. This effectively reduces the number of "null" observations to 10000, which makes this label still the most dominating in numbers, but not overwhelming. After the subsampling, the breakdown of the labels is shown in Table 4.

Label	eyem	chew	shiv	elpp	musc	null	total
Occurrences	7471	2727	1338	2663	4892	10763	29854
Percentage (%)	25.03	9.13	4.48	8.92	16.39	36.05	100

Table 4: Occurrences and Percentage of Each Label in Subsampled Dataset

Using this newly created dataset, the model was retrained until 10 epochs, and the validation accuracy increased until 33% in the first 2 epochs and fluctuated around 33% for the rest of the epochs indicating that the model's complexity is not sufficient enough to do the task.

4.3.2 Version 1: Recurrent Neural Network Approach

Using the prior knowledge that EEG signals are temporal, and previous works on detecting artifacts relied on statistical significances of various signal

features such as mean and standard deviation, the recurrent neural network seems to be the logical choice for the replacement of a network of 2 fully connected layers. After trying out different combinations of recurrent layers including SimpleRNN, Gated Recurrent Unit, Long Short-Term Memory (LSTM) layer was found out to be the most successful. As one can predict the behavior of the training the model from just observing first few epochs in general, different models have been compared by how much training loss was reduced in 3 epochs, and how much validation loss was reduced as a result of those 3 epochs. For the cases in which the loss function for this data set did not decrease significantly (by 0.1 or more), the losses never decreased in a reasonable time, and the model tended to overfit to the training data. The final model that was decided is organized in Table 5. The total number of trainable parameters is 117549, and this translates to 225KB of weights when weights are saved.

Layer (type)	Output Shape	Param #
Input_1 (InputLayer)	(None, 22, 250)	0
Lstm_1 (LSTM)	(None, 50)	60200
Dense_1 (Dense)	(None, 1024)	52224
Dense_2 (Dense)	(None, 5)	5125

Table 5: Model Structure for the RNN Based Classifier

The LSTM layer is to extract the temporal information embedded in the signal. The final dense layers are to do the classification tasks. The parameters on

each layer were chosen such that the model is as light as possible without sacrificing significant performance degradation. For the number of cells in the LSTM layer, a varying number of cells was tried such as 100, 200, 250, and the increasing the number of cells decreased the performance by overfitting. The model was trained on the training data using categorical cross-entropy as the loss function. The model was optimized using Adam optimizer with default learning rate and beta values. The batch size was 32, and the model was trained for 100 epochs. Each epoch takes about 40 seconds, and the training roughly took about half an hour. The result of this model will be given in the next chapter.

4.3.3 Version 2 Convolutional Neural Network Approach

Another approach chosen was using convolutional neural networks. As all the montages are available and ordered such that the arrangement reflects the actual spatial closeness of the electrodes, the conjecture was that there will be certain localities across different channels. As EEG measures net neural activity, if an area of the brain gets triggered, all the electrodes that are near that area will be triggered, which makes montages that are close in the ordered list will have similar activity. As convolutional neural networks are known to work well with image data by using the fact that in images pixels that are related are close together, it seems possible that convolutional layers will also work well with this

task. As there is only one-dimensional information available per time frame, 1-D convolutional layers were used instead of 2-D convolutional layers.

While convolutional layers capture the spatial information, the max-pooling layers capture the temporal information by grouping up time frames together. Extracting spatial information, and temporal information are done multiple times so that any hidden information can be extracted. Before max-pooling layers, batch normalization layers are added so that the values of latent space representation of the input signals are normalized and scaled. Parameter changes in layers during the training cause the layers to yield different outputs each iteration. This forces all the layers to readjust to the new distribution of the outputs every iteration, which delays the training. Batch normalization layer normalizes the activations to reduce these internal covariate shifts to make the training process to be faster, and more stable, especially for deep and large neural networks [32]. Finally, the model uses fully connected layers to do the classification task. Two versions of deep convolutional neural network models have been constructed. One version is deeper than the other one to see whether adding more layers helped with the classification or not. The structures of both versions are organized in Table 6 and Table 7.

Both versions were optimized using the Adam optimizer [31] with the default setting. The batch size was 32, and the model was trained for 30, and 100

epochs respectively. The first CNN model was highly overfitting to the train set at around epochs 40, as the validation loss went up by 10 times. The source of this behavior could not be tracked, so the number of epochs that the shallow CNN model was trained for was decreased to 30 epochs. The shallow CNN model took about 20 seconds per epochs, and the deeper model took about 40 seconds per epochs.

4.3.4 Ensemble Method

In addition to all the methods with different approaches, the final method that incorporates all the models was created. This model takes in the logit outputs of each of the 3 models, and simply adds the logits to do the decision making. Different combinations of models were tested, but the model that combines all three models, the RNN, the shallow CNN, the deep CNN, had the highest accuracy.

For all the models, binary classification versions were constructed to examine how well models detect artifacts. The binary classification task for this problem is determining whether a 1-second segment contains an artifact or not, which will be denoted as either “artifact” or “null”. The only deviation for these new models from the original models is the last dense layer. Instead of returning a label of length 5, the binary classification versions return the output label of length 2, (artifact, null). This causes the parameter numbers to be multiplied by 2/5 on the last dense layer. The number of total trainable parameters for the

shallow CNN classifier is 4728269, and for the deeper CNN classifier is 11548141. When weights are saved, the shallow CNN classifier requires 18.0MB while the deep CNN classifier requires 44.1MB. The results for both versions are given in the following chapter. All the construction of models and pipelines for input and output for the EEG signals are done in python, and the appropriate code sample is located in Appendix A.1.2.

Layer (type)	Output Shape	Param #
Input_1 (InputLayer)	(None, 22, 250)	0
conv1d_1 (Conv1D)	(None, 16, 250)	1072
batch_normalization_1	(None, 16, 250)	1000
max_pooling1d_1	(None, 16, 125)	0
conv1d_2 (Conv1D)	(None, 32, 125)	1568
batch_normalization_2	(None, 32, 125)	500
max_pooling1d_2	(None, 32, 63)	0
conv1d_3 (Conv1D)	(None, 64, 63)	6208
batch_normalization_3	(None, 64, 63)	252
max_pooling1d_3	(None, 64, 32)	0
conv1d_4 (Conv1D)	(None, 128, 32)	24704
batch_normalization_4	(None, 128, 32)	128
max_pooling1d_4	(None, 128, 16)	0
conv1d_5 (Conv1D)	(None, 256, 16)	98560
batch_normalization_5	(None, 256, 16)	64
max_pooling1d_5	(None, 256, 8)	0
conv1d_6 (Conv1D)	(None, 512, 8)	393728
batch_normalization_6	(None, 512, 8)	32
flatten_1	(None, 4096)	0
dense_1(Dense)	(None, 1024)	4195328
dense_2(Dense)	(None, 5)	5125

Table 6: Model Structure for the Shallow CNN Classifier

Layer (type)	Output Shape	Param #
Input_1 (InputLayer)	(None, 22, 250)	0
conv1d_1 (Conv1D)	(None, 16, 250)	1072
batch_normalization_1	(None, 16, 250)	1000
max_pooling1d_1	(None, 16, 125)	0
conv1d_2 (Conv1D)	(None, 32, 125)	1568
batch_normalization_2	(None, 32, 125)	500
max_pooling1d_2	(None, 32, 63)	0
conv1d_3 (Conv1D)	(None, 64, 63)	6208
batch_normalization_3	(None, 64, 63)	252
max_pooling1d_3	(None, 64, 32)	0
conv1d_4 (Conv1D)	(None, 128, 32)	24704
batch_normalization_4	(None, 128, 32)	128
max_pooling1d_4	(None, 128, 16)	0
conv1d_5 (Conv1D)	(None, 256, 16)	98560
batch_normalization_5	(None, 256, 16)	64
max_pooling1d_5	(None, 256, 8)	0
conv1d_6 (Conv1D)	(None, 512, 8)	393728
batch_normalization_6	(None, 512, 8)	32
max_pooling1d_6	(None, 512, 4)	0
conv1d_7 (Conv1D)	(None, 1024, 4)	1573888
batch_normalization_7	(None, 1024, 4)	16
max_pooling1d_7	(None, 1024, 2)	0
conv1d_8 (Conv1D)	(None, 1024, 2)	3146752
batch_normalization_8	(None, 1024, 2)	8
conv1d_9 (Conv1D)	(None, 1024, 2)	3146752
batch_normalization_9	(None, 1024, 2)	8
flatten_1	(None, 2048)	0
dense_1(Dense)	(None, 1024)	2098176
dense_2(Dense)	(None, 1024)	1049600
dense_3(Dense)	(None, 5)	5125

Table 7: Model Structure for the Deep CNN Classifier

5. Results

After optimizing hyperparameters, and model structures using validation set accuracy, each model was tested using the test set. For the first step of developing the model, the recurrent neural network model was trained for 100 epochs, on the data with the shivering label included. The confusion matrix for this model is shown in Figure 6.

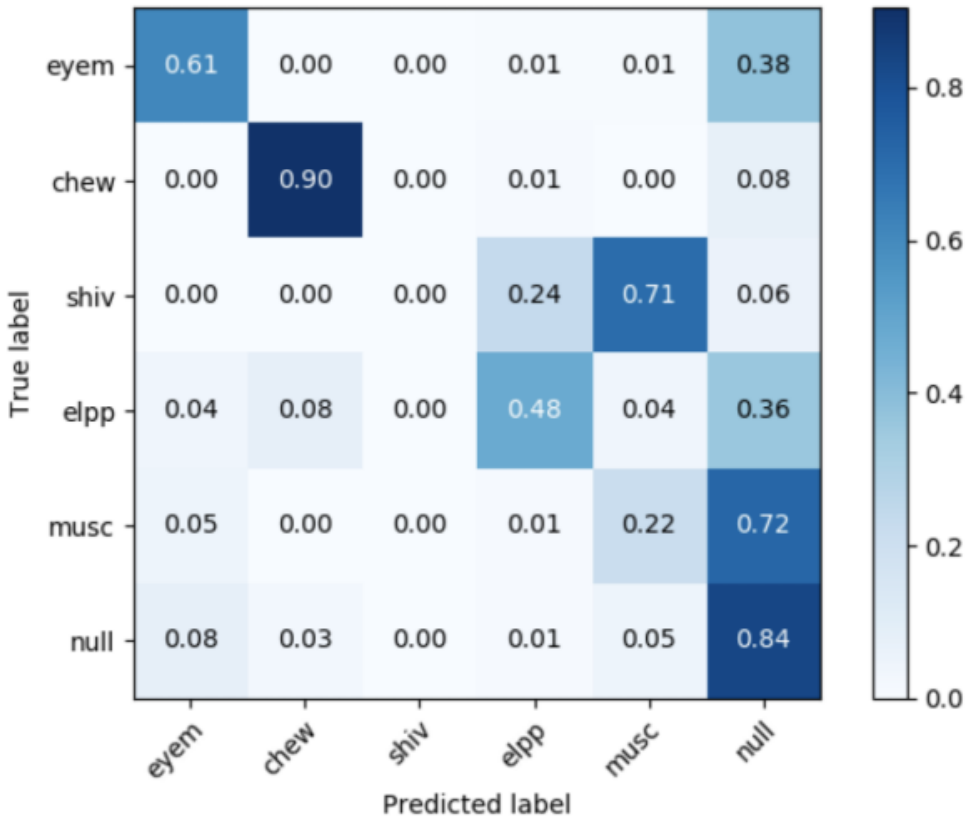

Figure 6: Confusion Matrix of RNN Based Model with All the Labels

The model completely fails to classify the shivering label and predicts all the shivering events to be either muscle movement event, electrode pop events, or the null event. In fact, the model does not predict anything to be the shivering event. The reason why the model failed to do so was there was no shivering label available in the training set, which caused the model to never be exposed to the label. As the number of shivering labels was so low as mentioned in the previous section, this label has been discarded.

From here on, all the models were trained and tested on the modified dataset that does not have the shivering label. The recurrent neural network model that was trained for 100 epochs. At the end of the training, the train set accuracy was 0.7168, and the validation accuracy was 0.4262. However surprisingly, the test set accuracy was 0.5801, and the confusion matrix is shown in Figure 7. The model does really well on predicting eye movement, and predicting "null". However, the model cannot predict electrode popping "elpp", and muscle movement "musc" that well. Unfortunately, this pattern persists in all the results. My conjecture of the behavior of the model is that eye movement and chewing labels have certain localities. Eye movement causes neurons in certain specific regions in the brain to fire, and chewing causes neurons in other specific regions in the brain to fire. This causes specific montages to be affected while leaving other montages to be like "null". As there is a distinguishing feature to be extracted consistently across all the patients, the model does well on eye

movement and chewing labels. However, for the cases of electrode popping, and general muscle movement, the region of the montages that will be affected is ambiguous. Electrode popping causes similar noise pattern to occur when it occurs, but this can be anywhere, and similar observation could be made regarding muscle movement.

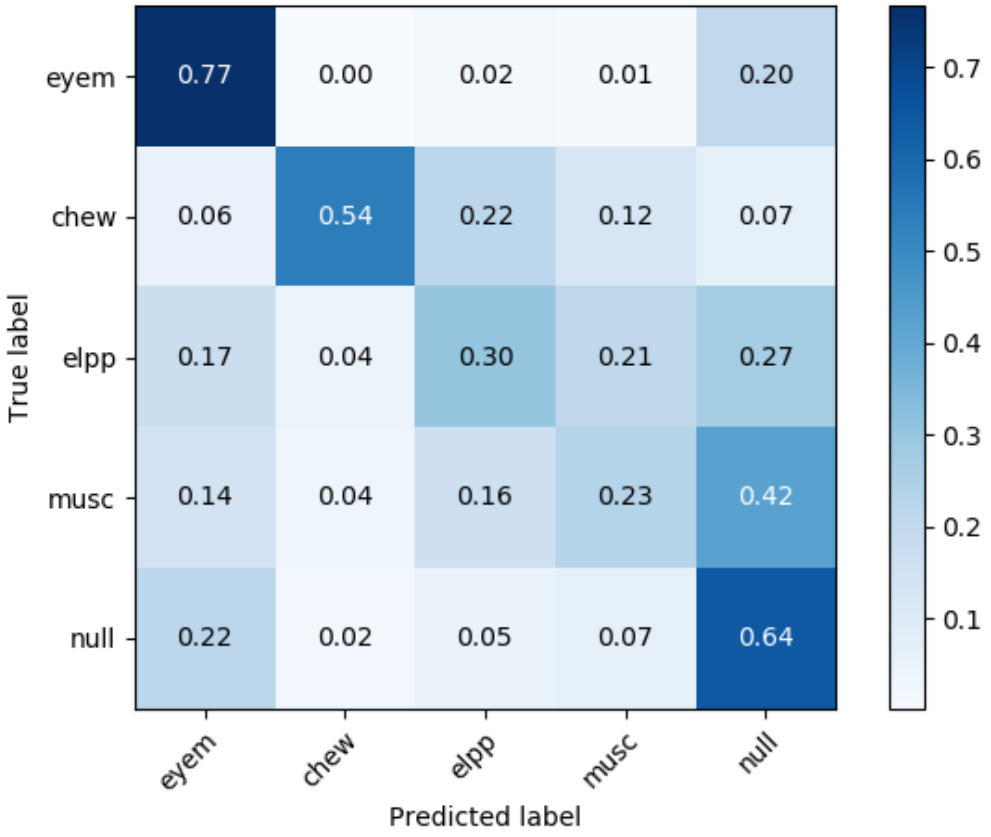

Figure 7: Confusion Matrix of RNN Based Model on Test Data

As limitations of precisely predicting the label are evident in the confusion matrix, at least, whether the model can act as an indicator for artifact presence needed to be tested. Hence, the models were retrained to do binary classification with the same number of epochs, and optimizer settings. The RNN based model trained to the train set accuracy of 0.9885, with validation accuracy of 0.6254. When tested on the test set, the highest accuracy was 0.7126.

However, this evaluation depends highly on the threshold that is set for detection. For example, when there are many examples of "null", or no artifacts, high accuracy could be achieved by intentionally raising the threshold of detection for artifacts high so that most of the examples are classified as "null". Then the system will have high accuracy while failing to act as a respectable classifier for artifacts. In order to evaluate the performance of the detection systems receiver operating characteristic (ROC) curves are used, which illustrate the ability of the systems to diagnose with different thresholds. The ROC curve plots the probability of detection versus the probability of false alarm [33]. The probability of detection which is also known as the true positive rate (TPR), sensitivity, or recall, denotes the proportion of actual positives that are correctly identified. Using the problem of this thesis as an example, the true positive rate is the proportion of segments that actually contain artifacts that are correctly classified by the model among all the segments that contain artifacts. The probability of false alarm, which is often referred to as the fall-out, the Type I error, or the false

positive rate (FPR), denotes the proportion of negatives that are misidentified as positives. Using this task as an example again, the false positive rate would be the proportion of segments that do not contain artifacts that are classified as containing artifacts by the model. A perfect classifier has the true positive rate of 1.0 and the false positive rate of 0.0, which makes the ROC curve to pass the upper left corner. Hence, a ROC curve that closely approaches the upper left corner indicates a system that discriminates well [34]. To numerically compare the performance of different ROC curves, the area under the curve (AUC) is computed to indicate how close the ROC curve is to the upper left corner. For all the ROC curves provided in this thesis, the area under the curve is also computed and provided.

In Figure 8, the ROC curve for the RNN based model is shown to visualize the performance of the system. The orange line is the ROC curve, and the dotted blue line is the straight line connecting the (0,0), and (1,1) points. The straight line indicates the worst possible detection system. At around the false positive rate, which indicates that the model predicts that the artifact exists when it does not, of 0.424, the true positive rate, which indicates that the model correctly predicts the presence of the artifact is 0.800. This indicates that the model would work in a system roughly, but would not be recommended in any device that requires high accuracy. The area under the curve is 0.75.

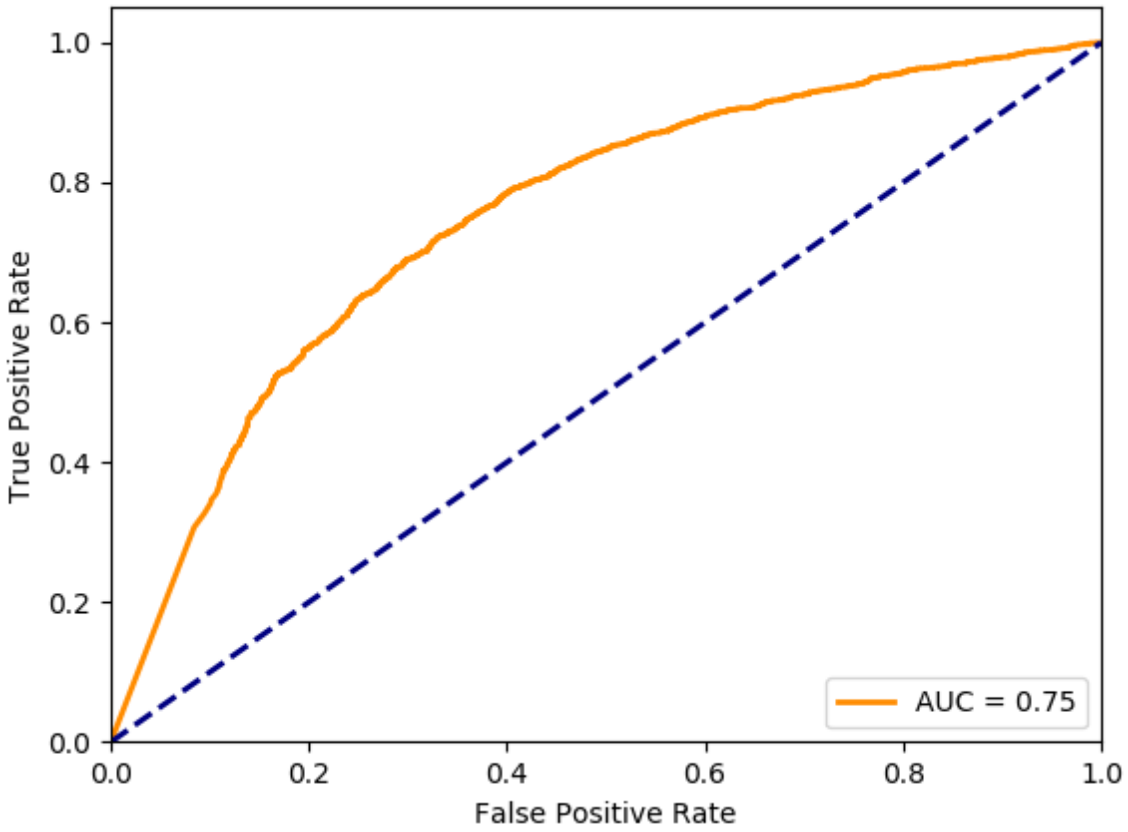

Figure 8: ROC Curve for RNN Based Model

Similar evaluations were done on the shallow CNN model and the deeper CNN model. The confusion matrices are shown in Figure 9, Figure 10, and ROC curves are shown in Figure 11, Figure 12.

The shallow CNN model was trained for 30 epochs, due to its tendency to overfit when it was trained for more than 40 epochs. The model was trained until the train accuracy of 0.7409, and the validation accuracy of 0.4203. The final test accuracy was 0.6515. Given that both RNN based model and CNN based model

trained till the validation accuracy was around 0.42, the fact that CNN based model did about 7% better in predicting the 5 class classification problem was interesting. One possibility is that the difference in the complexities of both models causes such difference. Comparing the number of trainable parameters, the CNN based model is 4 times bigger, and this may have helped the model to generalize better. However, as evident in Figure 9, this model does significantly better in predicting eye movement and chewing than electrode pops, and muscle movements.

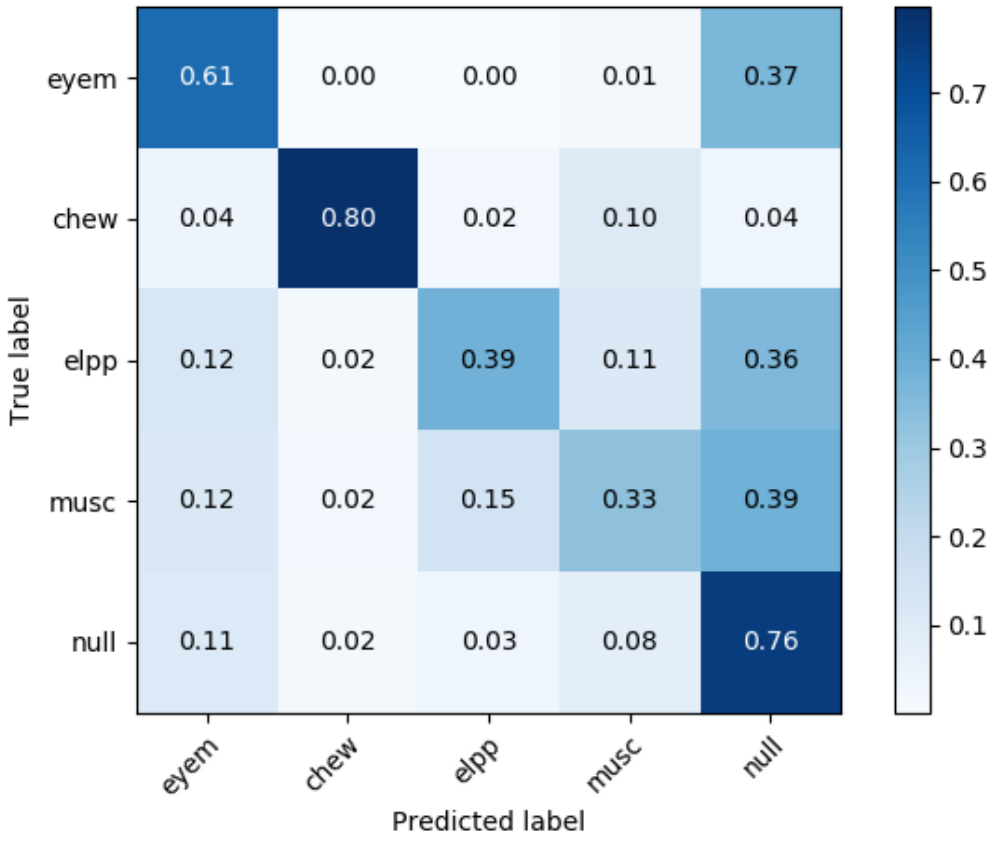

Figure 9: Confusion Matrix of the Shallow CNN Model

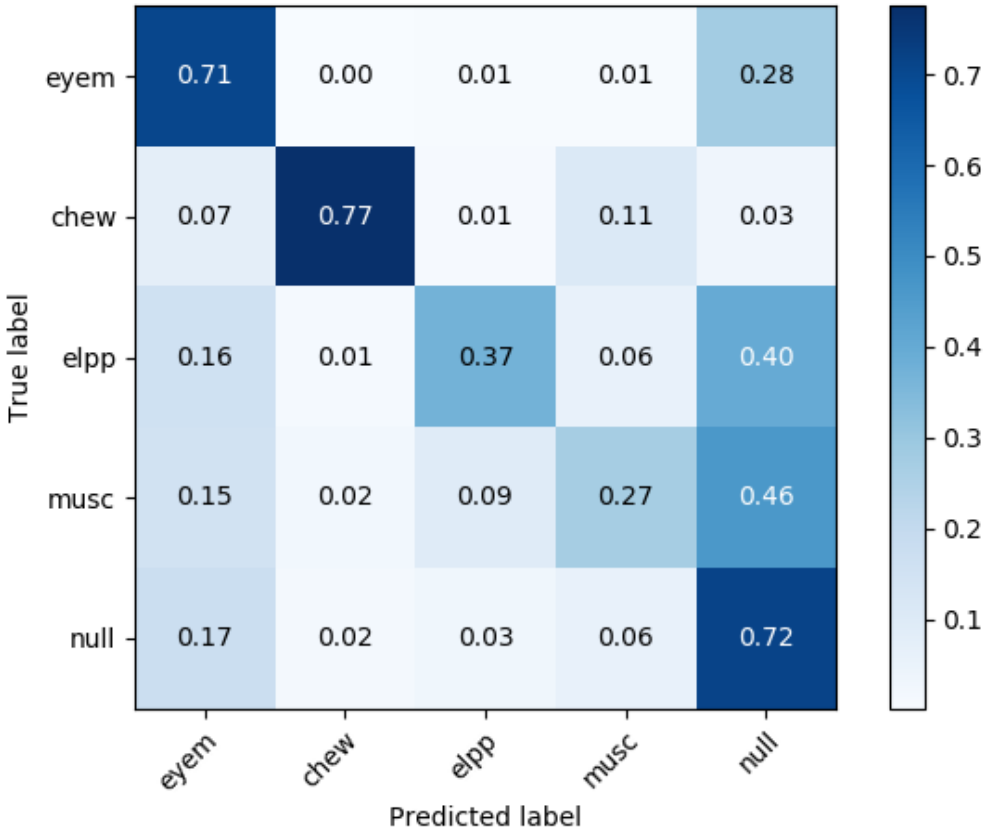

Figure 10: Confusion Matrix of the Deep CNN Model

The result for the deep CNN model is similar. The model was trained to the 100th epochs, and the train accuracy of 0.9472 was reached, and the validation accuracy at this epoch was 0.4430. This validation accuracy is slightly higher than the shallow CNN model. The final test accuracy is 0.6517, which is 0.0002 higher than the shallow CNN model. The confusion matrix shown in Figure 10, indicates similar behavior compared to other models. Hence, one can conclude that CNN based models work better in multi-class models, but RNN based model is much

lighter, and also simply making CNN based models more complex does not improve the performance of the model significantly.

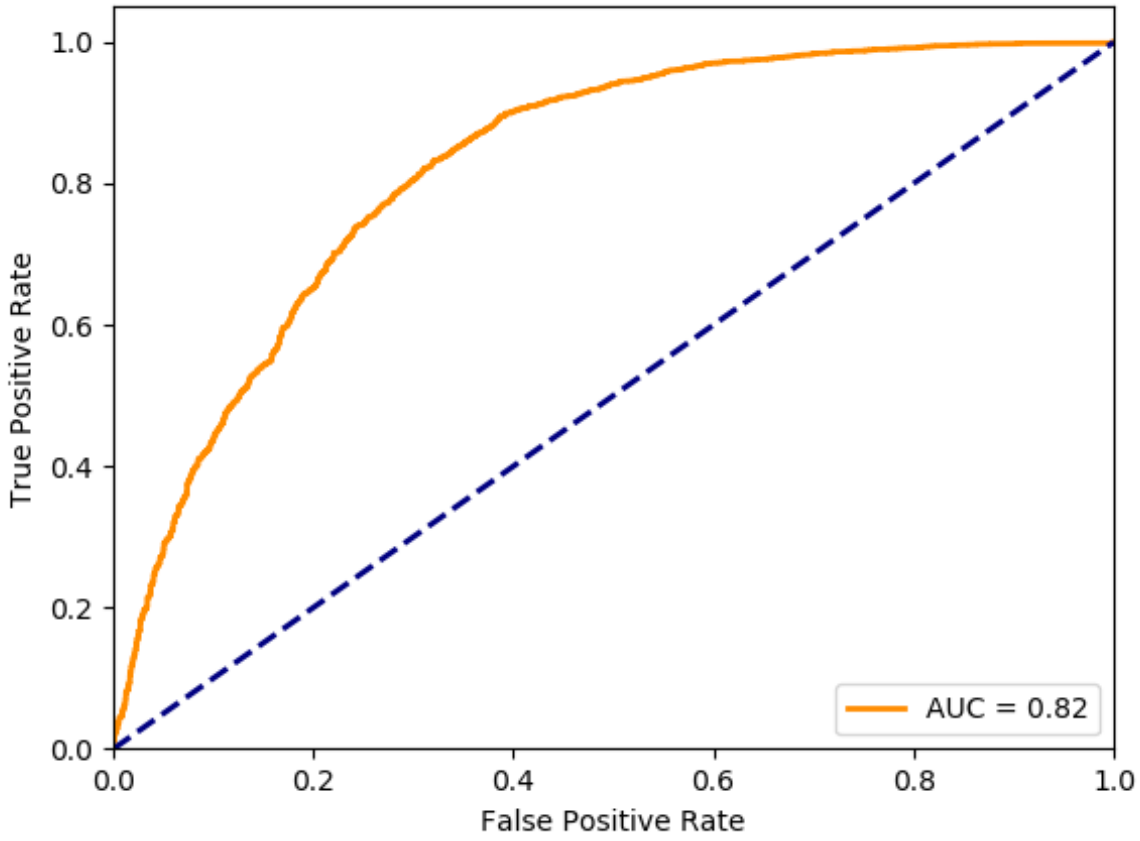

Figure 11: ROC curve for the Shallow CNN Model

The more interesting findings are ROC curves. The same analytic method was applied to both CNN based models just as in the RNN based model. The shallow CNN model was retrained for 30 epochs, and the deeper CNN based model

was retrained for 100 epochs. The train set accuracies were 0.8108, and 0.9684, the validation accuracies were 0.5227, 0.6008, and the test accuracies were 0.6958, and 0.7499 for the shallow and the deep CNN models respectively. Although these numbers might be misleading as the accuracy depends on the threshold of the binary classifier, but for the binary classification problem, the more complex and deeper model has a performance improvement of about 0.05. The receiver operating characteristic curves of CNN based models are shown in Figure 11, and Figure 12.

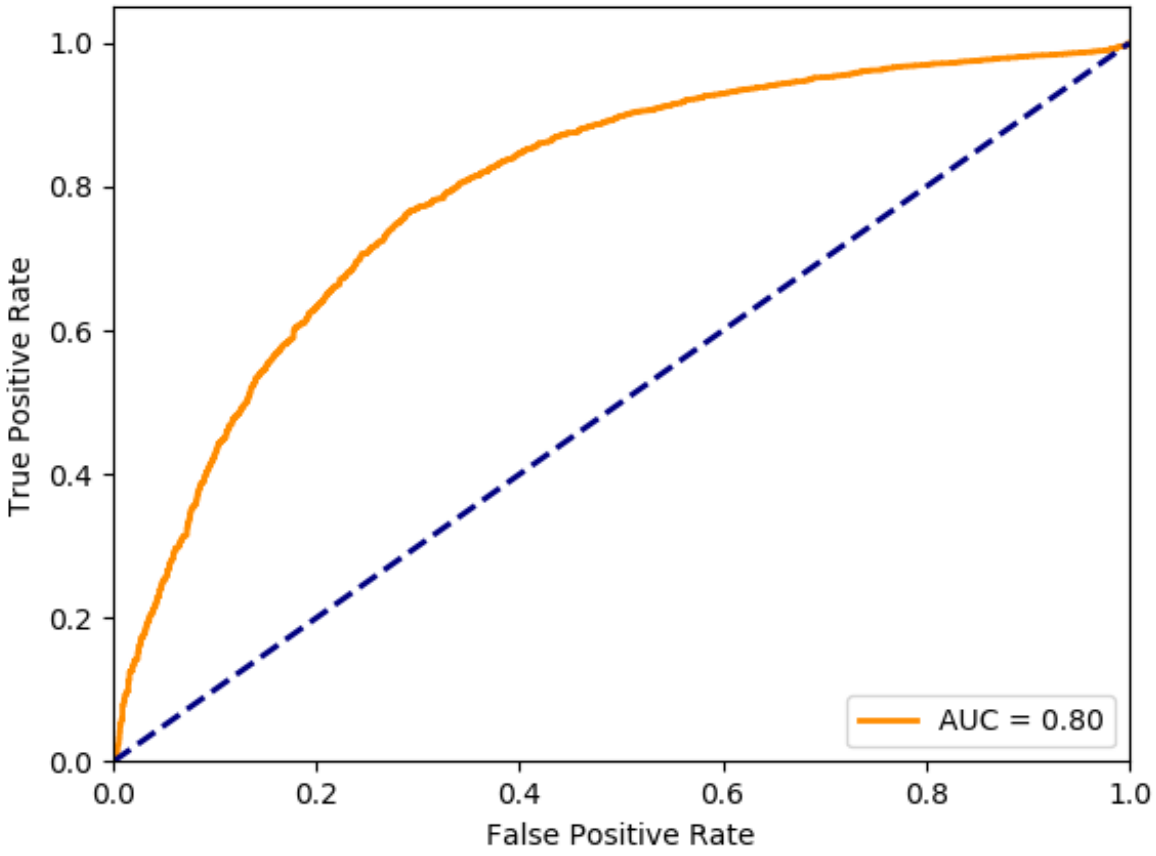

Figure 12: ROC Curve for the Deep CNN Model

These ROC curves, compared to that of the RNN based model, have a significantly higher area under the curve, indicating that it performs better. Numerically, the areas under the curve for the shallow CNN model and the deep CNN model are 0.82 and 0.80, respectively, which are larger than that of the RNN based model. At the true positive rate of 0.800, the false positive rates were 0.424, 0.295, and 0.339 for the RNN the shallow CNN, and the deep CNN models respectively. This indicates that CNN based models can predict the presence of artifact correctly, with much fewer false alarms compared to the RNN based model.

Lastly, the ensemble method was examined in the same procedure. The method incorporates all the other methods. The ensemble method simply sums the logits produced at the output layers of the other methods. The confusion matrix is shown in Figure 13. The ensemble method's accuracy measures are higher compared to all the other methods, except for the "musc" label. The shallow CNN model achieves the accuracy of 0.33 on the "musc" label, while the ensemble method achieves 0.28. Regardless, the ensemble method achieves the overall accuracy of 0.6759, which is the highest among all the methods. In addition to the confusion matrix, the ROC curve for the binary classification version of the model is produced. The ROC curve is shown in Figure 14, with all the ROC curves from other models for better comparison.

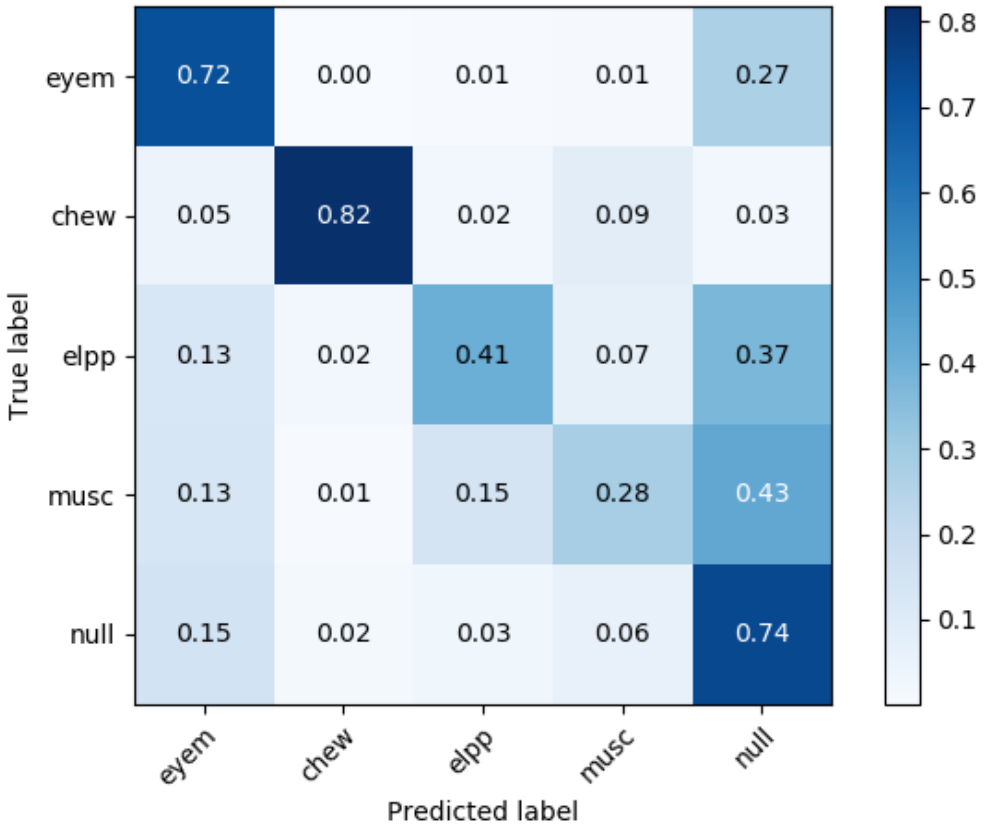

Figure 13: Confusion Matrix of the Ensemble Method

Interestingly the ROC curve for the shallow CNN model has the similar area under the curve as the ensemble method. The shallow CNN model has higher true positive rates in certain regions compared to the ensemble method, and the ensemble method performs superior to the shallow CNN model in the regions of lower threholds.

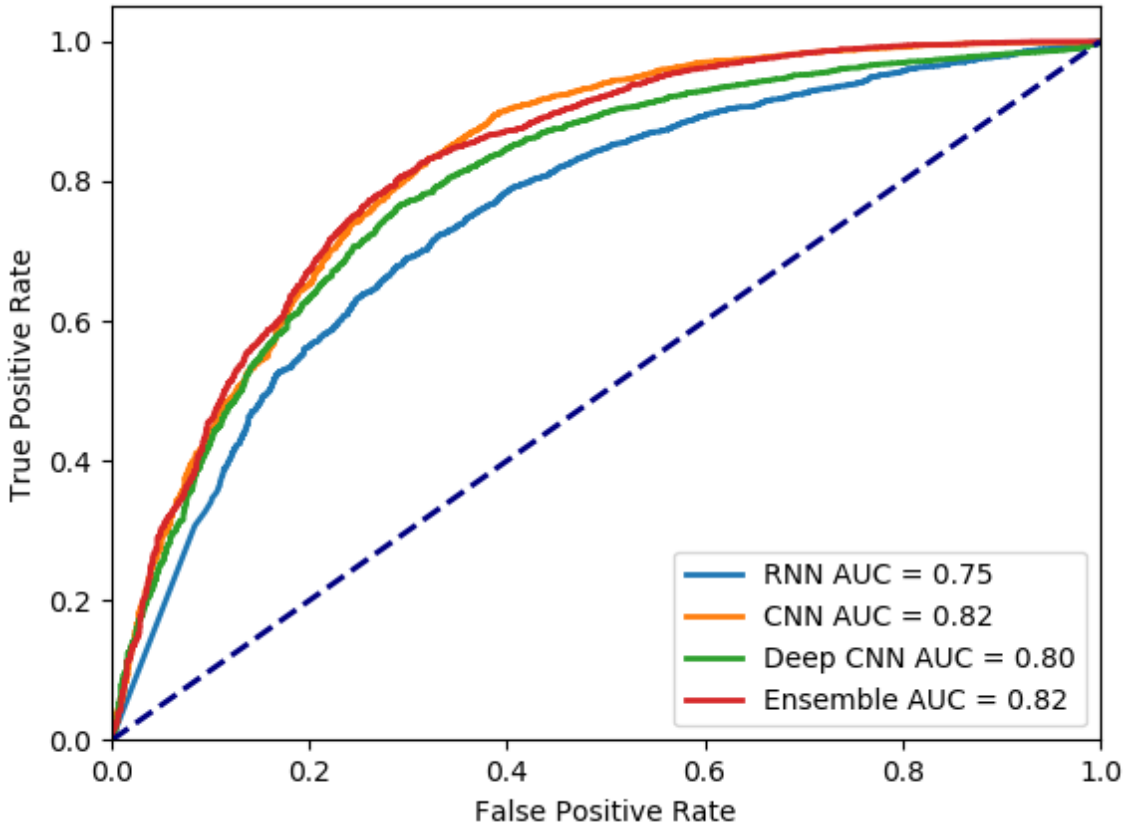

Figure 14: ROC Curves for All the Models

Because for the binary classification problem, the main purpose is to accurately point out the artifact events, the time-lapse system was proposed to further enhance the performance. The idea comes from the fact that often artifacts come in bursts, previous segment's label correlates well with the new segment that follows. This method does not change any of the models but rather works directly on the logits produced by the models. A sliding window adds all the logits in the window to produce a new logit that the classifier uses. Different methods of

producing the new logit were tried such as taking the maximum or doing weighted sum of the logits, but simply adding all the logits worked the best. Different sizes of sliding windows were tried, ranging from 1 to 10, but a 2-second window produced the best result. The ROC curves for the highest performing window setting are shown in Figure 15.

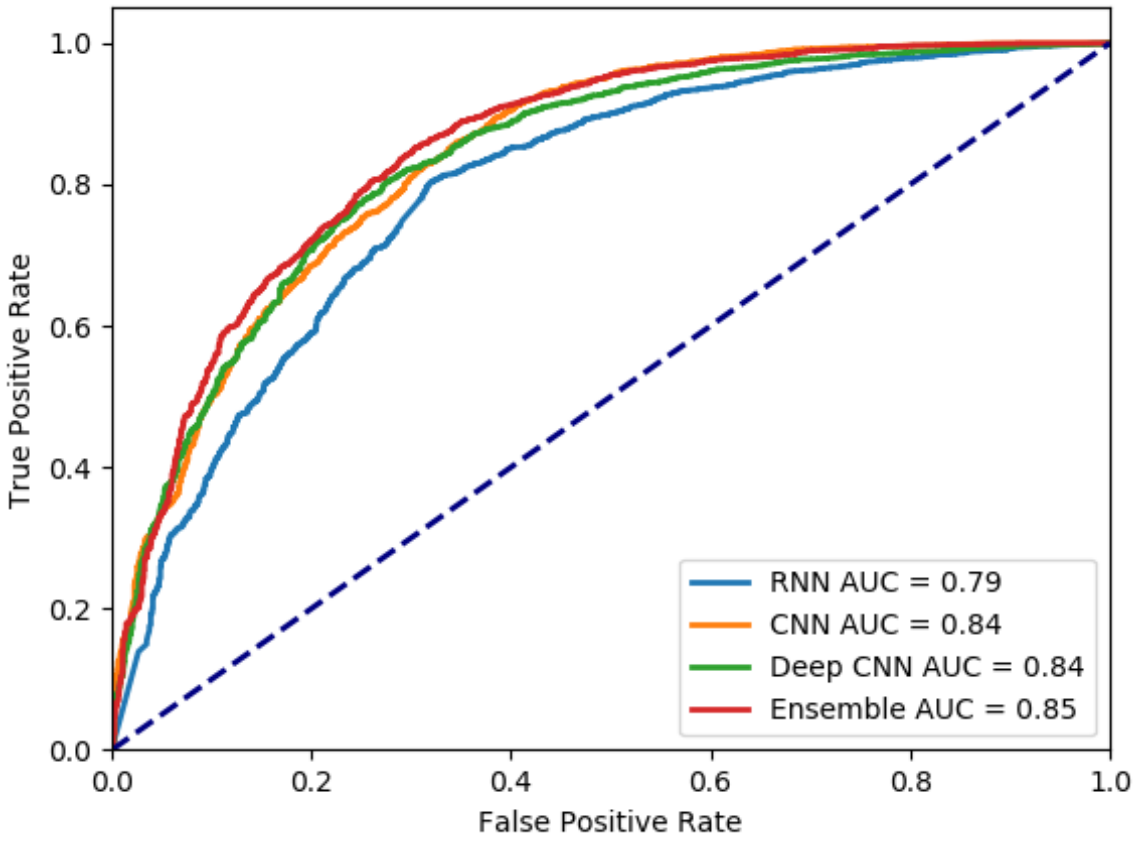

Figure 15: ROC Curves for All the Models with The Time-Lapse Method

The time-lapse method improves all the ROC curves, especially lifting the regions in the lower false positive rates. At the true positive rate of 0.800, the new time-lapse method yields the false positive rates of 0.310, 0.288, 0.268, 0.258 for RNN, shallow CNN, deep CNN, and ensemble methods respectively. This is a slight improvement from the false positive rate of 0.295 from the shallow CNN model without the sliding window. The ensemble method does the best for this method proposed.

In order to see the viability of the model in a real life settings, all the binary classification models were tested on a test set that contains all the "null" information without subsampling. The 5-class classification accuracies of the models are 0.7234, 0.7612, 0.7534, and 0.7808, for the RNN model, the shallow CNN model, the deep CNN model, and the ensemble model respectively. Only one confusion matrix from the best result is shown in the thesis as all the confusion matrices behave similarly. The resulting confusion matrix is shown in Figure 16. The increase in the accuracy comes from the fact that there are more "null" labels in the data set, hence the accuracy converges to the accuracy of predicting "null" label which is around 0.78 for the ensemble method.

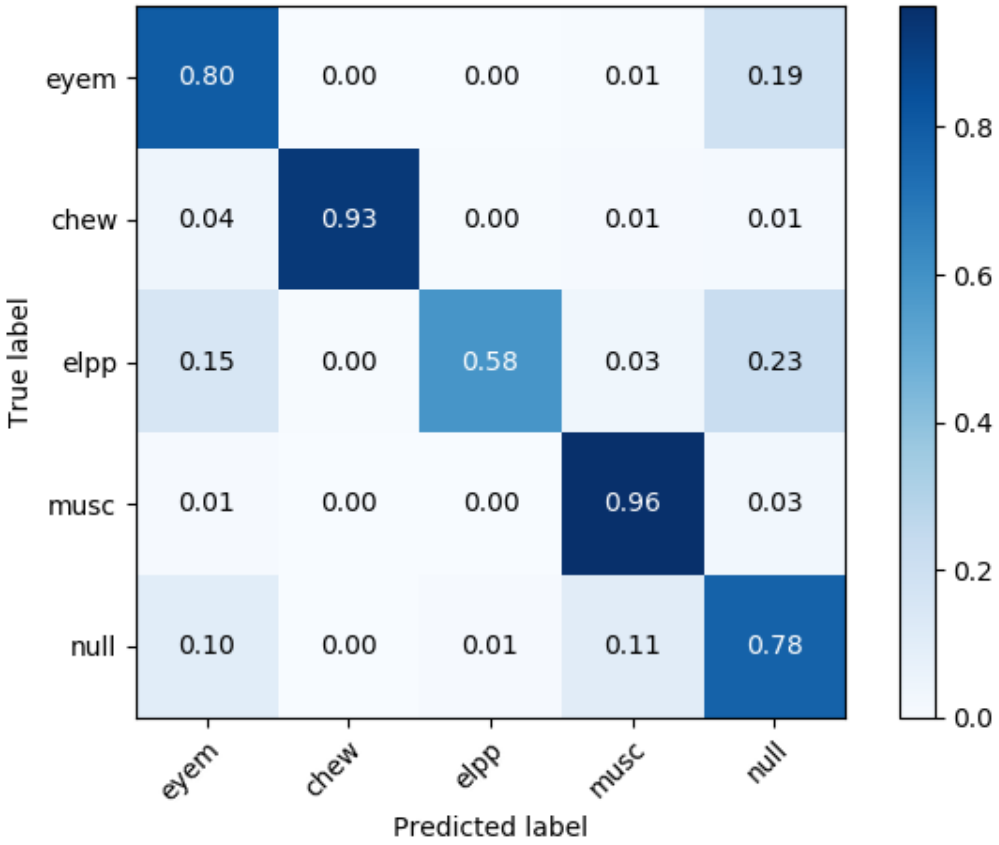

Figure 16: Confusion Matrix of the Ensemble Method on Original Data

The ROC curves for the binary classification problem using all the models on the non-subsampled data are shown in Figure 17 and Figure 18. The area under the curves are significantly higher than that of the subsampled data cases. These curves indicate the viability of the models in a real clinical settings.

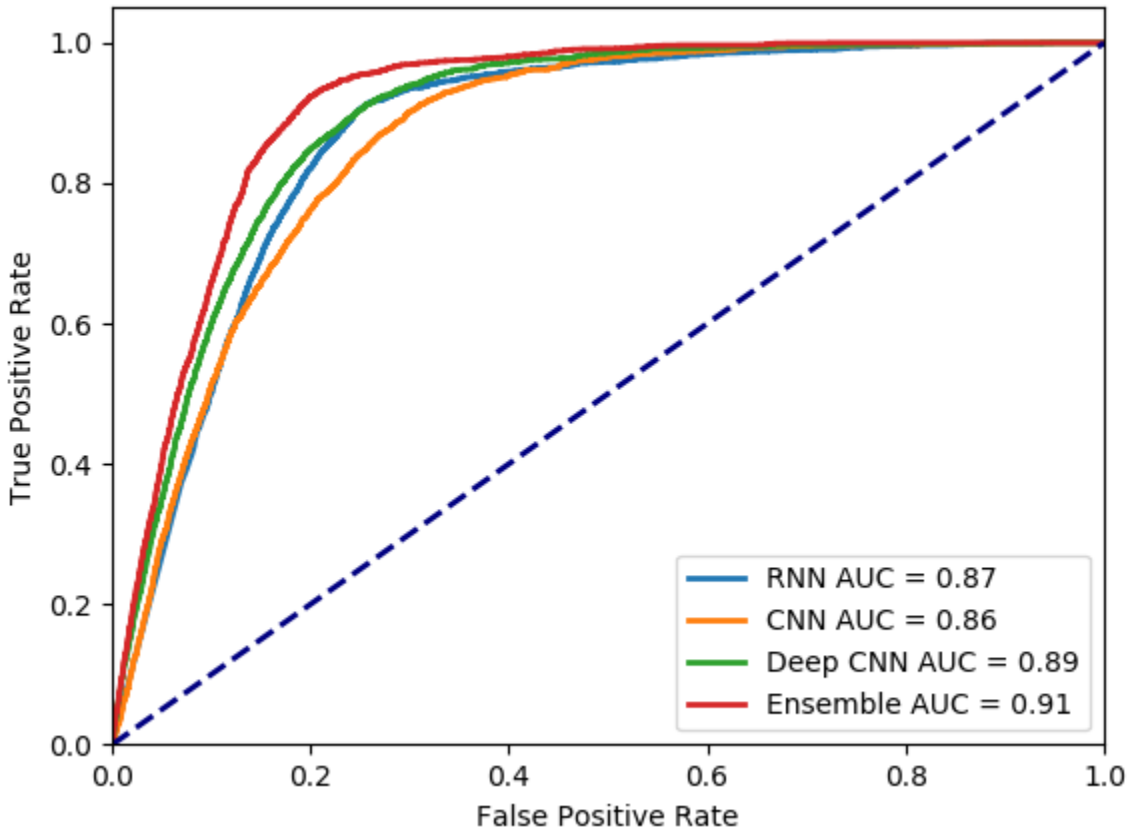

Figure 17: ROC Curves for All the Models on Original Data

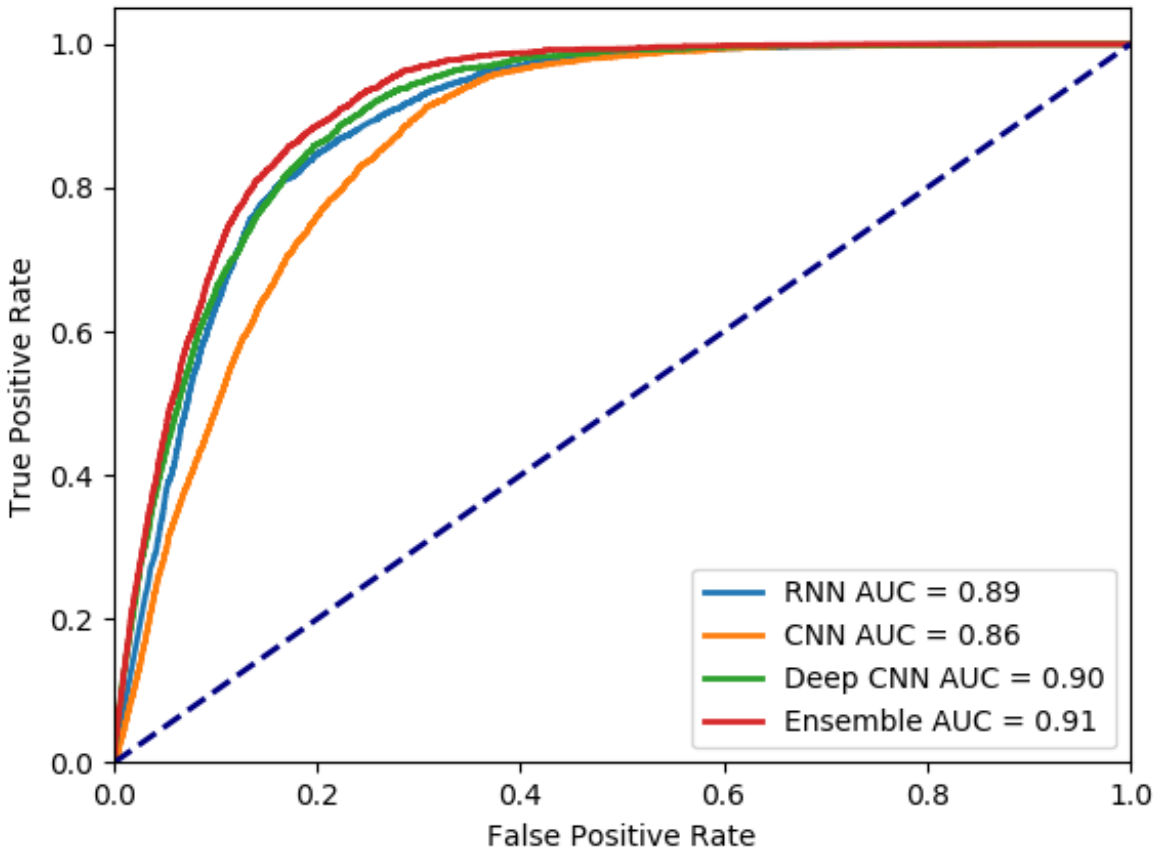

Figure 18: ROC Curves for All the Models with The Time-Lapse Method on Original Data

Lastly, average time elapsed in processing one example was computed for each model, for each classification problem to see whether the model is feasible for doing an online signal processing task of indicating whether the artifact exists or not. For reference, there were 5797 observations in the test set. In addition, the time elapsed loading the tensorflow module, and libraries as well as loading the data were not accounted for. The results for accuracy with default thresholds,

which looks at the maximum confidence level of each label, average time elapsed, and the size of each model are shown in Table 8. All the test results on this table are from the subsampled test data.

All the average time elapsed for inference for all the models is less than 1ms, for each of the 1 second time windows. This indicates that the model is able to predict the presence and the kind of artifact quickly. In addition, the size of the models is small enough to be implemented in a Raspberry Pi, which could make this model highly portable. Since the original EEG signals were expressed in 16-bit floating point values, the model can be further compressed if all the parameters are converted to 16-bit floating points instead of 32-bit floating points. This compression will approximately half the size of the model, further improving the portability. All the evaluations were done in python, and the appropriate code sample is available in Appendix, A.1.3.

Model	Average Time Elapsed (ms/sample)	Test Set Accuracy (%)	Size of the Model (KB)
RNN	0.707	58.01	476
RNN-binary	0.677	71.26	464
CNN	0.483	65.15	18526
CNN-binary	0.468	69.58	18514
DeepCNN	0.595	65.17	45189
DeepCNN-binary	0.568	74.99	45177
Ensemble	N/A	67.59	64191

Table 8: Time Elapsed, Accuracy, and Size of each Model

6. Conclusion

The research proposes three types of deep learning based machine learning model that learns to distinguish artifacts from the real signal and classify artifacts. Three models, RNN based model, and two CNN based models of different depth have been constructed and compared. In addition, the ensemble model was created that utilizes all the other methods. The ensemble model, which has the best overall performance, achieves a 67.59% 5-class classification accuracy, and a true positive rate of 80% at the false positive rate of 25.82% for the binary classification problem. The models are light and fast enough to be implemented in a portable device, such as Raspberry Pi, as the ensemble model, which contains all the other models has 65MB worth of trainable parameters, and the slowest model, the RNN based model, only takes about 0.7 ms to perform prediction on a 1-second windowed EEG signal. The speed of the ensemble model has not been tested but given that the slowest component in the model occupies less than 0.1% of the segment implies there is no problem. As this model can successfully detect whether artifacts are present in collected signals quickly, and can tell what type of artifacts they are, physicians can use this device while collecting data to check whether the data that are being collected is free of artifacts or not. If the data are being affected by artifacts, physicians can quickly check which kind of artifact is present, and act in response to that artifact.

Clinicians indicate that a sensitivity, which is the true positive rate, of 95%, and specificity, the false positive rate, of below 5% to be the minimum requirement for clinical acceptance [5]. As none of the models achieve that guideline yet, there is much more investigation needed in optimizing the models. Hence for the future works, an investigation into incorporating different features that can be extracted quickly, and larger and more complex models to reach the recommended guideline can be done. In addition, since the models were trained, validated, and tested on the first version of the EEG artifact corpus which only consists of observations from 310 patients, in the near future when there are more data available, the model could be trained again to see whether the lack of data was part of the inadequate performance. Also, since classification within the artifacts, excluding the "null" label, seems to work at high accuracies evident from the confusion matrix, and the binary classification of artifacts can have arbitrarily high true positive rate, an investigation on two-step system seems to be another interesting path to take on. This research envisioned to have a portable device that can be used during data acquisition. Building a portable machine that runs these models to predict the presence of artifacts, and to classify the artifacts should be the next step. Finally, testing this machine in a real-life setting will be beneficial to see if the machine works and to see if there are additional adjustments and improvements to make.

7. References

[1] M. J. Aminoff, "Principles of Neural Science. 4th edition," *Muscle & Nerve,* vol. 24, no. 6, pp. 839-839, 2001.

[2] E. K. S. Louis, L. C. Frey, J. W. Britton, J. L. Hopp, P. Korb, M. Z. Koubeissi, W. E. Lievens and E. M. Pestana-Knight, Electroencephalography (EEG): An Introductory Text and Atlas of Normal and Abnormal Findings in Adults, Children, and Infants, 2016.

[3] J. DellaBadia, W. L. Bell, J. W. Keyes, V. P. Mathews and S. S. Glazier, "Assessment and cost comparison of sleep-deprived EEG, MRI and PET in the prediction of surgical treatment for epilepsy," *Seizure-european Journal of Epilepsy,* vol. 11, no. 5, pp. 303-309, 2002.

[4] I. Obeid and J. Picone, "The Temple University Hospital EEG Data Corpus," *Frontiers in Neuroscience,* vol. 10, p. 196, 2016.

[5] M. Golmohammadi, A. H. H. N. Torbati, S. L. d. Diego, I. Obeid and J. Picone, "Automatic Analysis of EEGs Using Big Data and Hybrid Deep Learning Architectures," *Frontiers in Human Neuroscience,* vol. 13, 2019.

[6] H. Nolan, R. Whelan and R. B. Reilly, "FASTER: Fully Automated Statistical Thresholding for EEG artifact Rejection," *Journal of Neuroscience Methods,* vol. 192, no. 1, pp. 152-162, 2010.

[7] T.-W. Lee, M. A. Girolami and T. J. Sejnowski, "Independent component analysis using an extended infomax algorithm for mixed subgaussian and supergaussian sources," *Neural Computation,* vol. 11, no. 2, pp. 417-441, 1999.

[8] B. Singh and H. Wagatsuma, "A Removal of Eye Movement and Blink Artifacts from EEG Data Using Morphological Component Analysis," *Computational and Mathematical Methods in Medicine,* vol. 2017, pp. 1861645-1861645, 2017.

[9] W. D. Clercq, A. Vergult, B. Vanrumste, W. V. Paesschen and S. V. Huffel, "Canonical Correlation Analysis Applied to Remove Muscle Artifacts From the Electroencephalogram," *IEEE Transactions on Biomedical Engineering,* vol. 53, no. 12, pp. 2583-2587, 2006.

[10] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016.

[11] Y. Roy, H. J. Banville, I. Albuquerque, A. Gramfort, T. H. Falk and J. Faubert, "Deep learning-based electroencephalography analysis: a systematic review.," *arXiv preprint arXiv:1901.05498,* 2019.

[12] V. Krishnaveni, S. Jayaraman, A. Gunasekaran and K. Ramadoss, "Automatic Removal of Ocular Artifacts using JADE Algorithm and Neural Network," *International Journal of Computer and Information Engineering,* vol. 2, no. 4, pp. 1330-1341, 2007.

[13] A. Hasasneh, N. Kampel, P. Sripad, N. J. Shah and J. Dammers, "Deep Learning Approach for Automatic Classification of Ocular and Cardiac Artifacts in MEG Data," *The Journal of Engineering,* vol. 2018, pp. 1-10, 2018.

[14] C. M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), Berlin, Heidelberg: Springer-Verlag, 2006.

[15] R. Benenson, "Classification datasets results," 22 2 2016. [Online]. Available: http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html. [Accessed 25 3 2019].

[16] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui,

L. Sifre, G. van den Driessche, T. Graepel and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature,* vol. 550, pp. 354--, October 2017.

[17] Glosser.ca, "File:Colored neural network.svg," Wikipedia, 28 February 2013. [Online]. Available: https://en.wikipedia.org/wiki/File:Colored_neural_network.svg. [Accessed 7 April 2019].

[18] C. Nwankpa, W. Ijomah, A. Gachagan and S. Marshall, "Activation Functions: Comparison of trends in Practice and Research for Deep Learning.," *arXiv preprint arXiv:1811.03378,* 2018.

[19] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of Physiology,* vol. 195, no. 1, pp. 215-243, 1968.

[20] Theano Development Team, "Convolutioinal Neural Networks (LeNet)," [Online]. Available: http://deeplearning.net/tutorial/lenet.html. [Accessed 8 April 2019].

[21] FirelordPhoenix, "File:MaxpoolSample2.png," computersciencewiki, 26 February 2018. [Online]. Available:

https://computersciencewiki.org/index.php/File:MaxpoolSample2.png. [Accessed 8 April 2019].

[22] X. Li and X. Wu, "Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015.

[23] Y. Bengio, P. Y. Simard and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks,* vol. 5, no. 2, pp. 157-166, 1994.

[24] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation,* vol. 9, no. 8, pp. 1735-1780, 1997.

[25] BruceBlaus, "File:Blausen 0657 MultipolarNeuron.png," Wikipedia, 30 September 2013. [Online]. Available: https://en.wikipedia.org/wiki/File:Blausen_0657_MultipolarNeuron.png. [Accessed 8 April 2019].

[26] A. F. Jackson and D. J. Bolger, "The neurophysiological bases of EEG and EEG measurement: A review for the rest of us," *Psychophysiology,* vol. 51, no. 11, pp. 1061-1071, 2014.

[27] R. Oostenveld and P. Praamstra, "The five percent electrode system for high-resolution EEG and ERP measurements," *Clinical Neurophysiology,* vol. 112, no. 4, pp. 713-719, 2001.

[28] トマトン 124, "File:21 electrodes of International 10-20 system for EEG.svg," Wikimedia Commons, 30 May 2010. [Online]. Available: https://commons.wikimedia.org/wiki/File:21_electrodes_of_International_10-20_system_for_EEG.svg. [Accessed 2 March 2019].

[29] F. L. d. Silva and E. Niedermeyer, Electroencephalography: Basic principles, clinical applications, and related fields, 1987.

[30] The McGill Physiology Virtual Lab, "Biomedical Signals Acquisition," The McGill Physiology Virtual Laboratory, [Online]. Available: https://www.medicine.mcgill.ca/physio/vlab/biomed_signals/eeg_n.htm. [Accessed 8 April 2019].

[31] D. P. Kingma and J. L. Ba, "Adam: A Method for Stochastic Optimization," *international conference on learning representations,* 2015.

[32] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *international conference on machine learning,* pp. 448-456, 2015.

[33] M. Richards, Fundamentals of Radar Signal Processing, 2005.

[34] M. H. Zweig and G. Campbell, "Receiver-operating characteristic (ROC) plots: a fundamental evaluation tool in clinical medicine.," *Clinical Chemistry,* vol. 39, no. 4, pp. 561-577, 1993.

A. Appendix

A.1 Code Samples

A.1.1 Data Preprocessing

```
1. import os
2. import pandas as pandas
3. import numpy as np
4. import pyedflib
5. import matplotlib.pyplot as plt
6. from scipy import signal
7. import random
8. from tqdm import tqdm
9.
10.   DIR_DATA = "D:/Data/Master/edf"
11.   DIR_SAVE = "D:/Data/Master_6/"
12.   DIR_SAVE_TRAIN = DIR_SAVE + 'train/'
13.   DIR_SAVE_TEST = DIR_SAVE + 'test/'
14.   DIR_SAVE_VAL = DIR_SAVE + 'val/'
15.
16.   null_ = np.zeros(6)
17.   null_[5] = 1
18.   null_.astype(np.float32)
19.   np.random.seed(31415)
20.   def file_import():
21.       train_ratio = 0.75
22.       test_ratio =  0.15
23.       val_ratio =   0.10
```

```
    edf_store = []
    method_store = []
    patient_store = []
    for r, d, f in os.walk(DIR_DATA):
        for file in f:
            if ".edf" in file:
                if "01_tcp_ar" in r:
                    edf_store.append(os.path.join(r,file))
                    patient_store.append(file[0:8])
                    method_store.append(1)
                elif "02_tcp_le" in r:
                    edf_store.append(os.path.join(r,file))
                    patient_store.append(file[0:8])
                    method_store.append(2)
    unique_ID = list(set(patient_store))
    train_index = int(round(len(unique_ID)*train_ratio))
    test_index = int(round(len(unique_ID)*test_ratio))
    random.shuffle(unique_ID)
    train_ID = unique_ID[0:train_index]
    test_ID = unique_ID[train_index:train_index+test_index]
    val_ID = unique_ID[train_index+test_index:]
    train_edf = []
    train_method = []
    for i in train_ID:
        for ii in range(len(edf_store)):
            if i in edf_store[ii]:
                train_edf.append(edf_store[ii])
                train_method.append(method_store[ii])
    test_edf = []
    test_method = []
    for i in test_ID:
```

```
            for ii in range(len(edf_store)):
                if i in edf_store[ii]:
                    test_edf.append(edf_store[ii])
                    test_method.append(method_store[ii])
        val_edf = []
        val_method = []
        for i in val_ID:
            for ii in range(len(edf_store)):
                if i in edf_store[ii]:
                    val_edf.append(edf_store[ii])
                    val_method.append(method_store[ii])
        return train_edf, train_method, test_edf, test_method, val_edf,
    val_method

    def info_retrievel(flag):
        if flag == 2: #LE
            return ['EEG FP1-LE', 'EEG FP2-LE', 'EEG F3-LE', 'EEG F4-
    LE', 'EEG C3-LE', 'EEG C4-LE', 'EEG P3-LE', 'EEG P4-LE', 'EEG O1-
    LE', 'EEG O2-LE', 'EEG F7-LE', 'EEG F8-LE', 'EEG T3-LE', 'EEG T4-
    LE', 'EEG T5-LE', 'EEG T6-LE', 'EEG CZ-LE','EEG A1-LE', 'EEG A2-LE']
        return ['EEG FP1-REF', 'EEG FP2-REF', 'EEG F3-REF', 'EEG F4-
    REF', 'EEG C3-REF', 'EEG C4-REF', 'EEG P3-REF', 'EEG P4-REF', 'EEG O1-
    REF', 'EEG O2-REF', 'EEG F7-REF', 'EEG F8-REF', 'EEG T3-REF', 'EEG T4-
    REF', 'EEG T5-REF', 'EEG T6-REF', 'EEG CZ-REF','EEG A1-REF','EEG A2-
    REF']

    def montage_form(file,method):
        label = file.getSignalLabels()
        montage = []
        needed = info_retrievel(method)
        FP1 = file.readSignal(label.index(needed[0]))
```

```
78.         FP2 = file.readSignal(label.index(needed[1]))
79.         F3 = file.readSignal(label.index(needed[2]))
80.         F4 = file.readSignal(label.index(needed[3]))
81.         C3 = file.readSignal(label.index(needed[4]))
82.         C4 = file.readSignal(label.index(needed[5]))
83.         P3 = file.readSignal(label.index(needed[6]))
84.         P4 = file.readSignal(label.index(needed[7]))
85.         N01 = file.readSignal(label.index(needed[8]))
86.         N02 = file.readSignal(label.index(needed[9]))
87.         F7 = file.readSignal(label.index(needed[10]))
88.         F8 = file.readSignal(label.index(needed[11]))
89.         T3 = file.readSignal(label.index(needed[12]))
90.         T4 = file.readSignal(label.index(needed[13]))
91.         T5 = file.readSignal(label.index(needed[14]))
92.         T6 = file.readSignal(label.index(needed[15]))
93.         CZ = file.readSignal(label.index(needed[16]))
94.         A1 = file.readSignal(label.index(needed[17]))
95.         A2 = file.readSignal(label.index(needed[18]))
96.         montage.append(FP1-F7)
97.         montage.append(F7-T3)
98.         montage.append(T3-T5)
99.         montage.append(T5-N01)
100.        montage.append(FP2-F8)
101.        montage.append(F8-T4)
102.        montage.append(T4-T6)
103.        montage.append(T6-N02)
104.        montage.append(A1-T3)
105.        montage.append(T3-C3)
106.        montage.append(C3-CZ)
107.        montage.append(CZ-C4)
108.        montage.append(C4-T4)
```

```
109.      montage.append(T4-A2)
110.      montage.append(FP1-F3)
111.      montage.append(F3-C3)
112.      montage.append(C3-P3)
113.      montage.append(P3-N01)
114.      montage.append(FP2-F4)
115.      montage.append(F4-C4)
116.      montage.append(C4-P4)
117.      montage.append(P4-N02)
118.      montage = np.asarray(montage,dtype = np.float32)
119.      return montage
120.
121.  def label_form(root):
122.      labels = ['eyem','chew','shiv','elpp','musc','null']
123.      directory = root.replace('.edf','.tse')
124.      lines = open(directory,'r').readlines()[2:]
125.      time = []
126.      label = []
127.      for l in lines:
128.          temp = l.split()
129.          temp2 = np.zeros(len(labels))
130.          temp2[labels.index(temp[2])] = 1
131.          temp2 == 1
132.          label.append(temp2)
133.          time.append(np.array([temp[0],temp[1]],dtype = np.float32))

134.      time = np.array(time,dtype = np.float32)
135.      label = np.array(label,dtype = np.float32)
136.      return label,time
137.
138.  def create_data(directory,method,subsample):
```

```
139.       SAMPLING_RATE = 250
140.       file = pyedflib.EdfReader(directory)
141.       montage = montage_form(file,method)
142.       label, time = label_form(directory)
143.       montage = signal.resample(montage,int(time[-1][-
   1]*SAMPLING_RATE),axis = 1)
144.       file._close()
145.       x = []
146.       y = []
147.       index = 0
148.       counter = 0
149.       for i in range(int(time[-1][-1])):
150.           if subsample == 1 and np.all(label[np.sum(i >= time[:,0])-
   1,:] == null_):
151.               counter += 1
152.               if counter == 30:
153.                   counter = 0
154.                   x.append(montage[:,i*SAMPLING_RATE:(i+1)*SAMPLING_RA
   TE])
155.                   y.append(label[np.sum(i >= time[:,0])-1,:])
156.           else:
157.               x.append(montage[:,i*SAMPLING_RATE:(i+1)*SAMPLING_RATE])

158.               y.append(label[np.sum(i >= time[:,0])-1,:])
159.       x = np.array(x,dtype = np.float32)
160.       y = np.array(y,dtype = np.float32)
161.       return x, y
162.
163.   def data_save(subsample = 1):
164.       train_edf, train_method, test_edf, test_method, val_edf, val_met
   hod = file_import()
```

```
    for i in tqdm(range(len(train_edf))):
        x,y = create_data(train_edf[i],train_method[i],subsample)
        name = str(i).zfill(3)
        np.save(DIR_SAVE_TRAIN+name,x)
        np.save(DIR_SAVE_TRAIN+name+'l',y)

    for i in tqdm(range(len(test_edf))):
        x,y = create_data(test_edf[i],test_method[i],subsample)
        name = str(i).zfill(3)
        np.save(DIR_SAVE_TEST+name,x)
        np.save(DIR_SAVE_TEST+name+'l',y)

    for i in tqdm(range(len(val_edf))):
        x,y = create_data(val_edf[i],val_method[i],subsample)
        name = str(i).zfill(3)
        np.save(DIR_SAVE_VAL+name,x)
        np.save(DIR_SAVE_VAL+name+'l',y)

if __name__ == '__main__':
    data_save(0)
```

A.1.2 Model

```
1. # Libraries
2. import data_pre_processing as dpp
3. import tensorflow as tf
4. import numpy as np
5. import numpy.matlib
6. import keras
7. from tqdm import tqdm
8. from keras.utils import Sequence
9. from keras import optimizers
10.    from keras.models import Model
11.    from keras.layers import Conv1D, Dense, Activation, MaxPooling1D, Fl
   atten, Reshape, BatchNormalization, LeakyReLU, Lambda, Input, Dropout,
   LSTM, Conv2D, MaxPooling2D
12.    from sklearn import preprocessing
13.    from sklearn.preprocessing import normalize
14.    import time
15.
16.    # Locations to use for save, and load
17.    DIR_SAVE = "D:/Data/Master_4/"
18.    # DIR_SAVE = "D:/Data/Master_6/"
19.    DIR_SAVE_TRAIN = DIR_SAVE + 'train/'
20.    DIR_SAVE_VAL = DIR_SAVE + 'val/'
21.    DIR_SAVE_TEST = DIR_SAVE + 'test/'
22.    NUM_OBSERVATION = [224, 23, 56] # Number of Observations for train,
   val, test
23.    mu = -0.8698222
24.    std = 196.00111
25.
26.    def data_load(dir,i):
```

```
27.         root = dir + str(i).zfill(3)
28.         x = np.load(root+'.npy')
29.         y = np.load(root+'l.npy')
30.         return x,y
31.
32.     def artifact_check(y):
33.         if y.shape[1] == 2:
34.             max = [0,0]
35.             min = [9999,9998]
36.         else:
37.             max = [0,0,0,0,0]
38.             min = [9999,9999,9999,9999,9999]
39.
40.         for i in range(y.shape[0]):
41.             if i ==0:
42.                 prev = y[i]
43.                 counter = 1
44.             else:
45.                 if np.array_equal(prev, y[i]):
46.                     counter+=1
47.                 else:
48.                     index = np.where(prev == 1)
49.                     index = int(index[0])
50.                     if (min[index] > counter):
51.                         min[index] = counter
52.
53.                     if (max[index] < counter):
54.                         max[index] = counter
55.                     counter = 1
56.                 prev = y[i]
57.         print(max)
```

```
        print(min)

    def load_all(directory, infos, b, time_lapse = 0):
        x,y = data_load(directory,0)
        k = []
        k.append(y.shape[0])
        for i in infos:
            if i == 0:
                continue
            else:
                temp1, temp2 = data_load(directory, i)
                x = np.concatenate((x,temp1),axis = 0)
                y = np.concatenate((y,temp2),axis = 0)
                k.append(temp2.shape[0])
        y = np.concatenate((y[:,0:2],y[:,3:6]),axis = 1)
        mask = np.sum(y,axis=1)
        y = y[mask != 0]
        x = x[mask != 0]
        if b == 1:
            y = np.concatenate((np.abs(y[:,4:5]-1),y[:,4:5]),axis = 1)
        # artifact_check(y)
        if time_lapse == 1:
            return x, y, k
        return x, y

    class DataGenerator(Sequence):
        def __init__(self,x,y,batch_size):
            self.num_ = y.shape[0]
            self.batch_size = batch_size
            self.x = x
            self.y = y
```

```
89.         def __len__(self):
90.             return int(np.ceil(self.num_/self.batch_size))
91.
92.         def __getitem__(self,index):
93.             if (index+1)*self.batch_size > self.num_:
94.                 in_ = self.x[self.num_-self.batch_size:]
95.                 out = self.y[self.num_-self.batch_size:]
96.             else:
97.                 in_ = self.x[index*self.batch_size:(index+1)*self.batch_
    size]
98.                 out = self.y[index*self.batch_size:(index+1)*self.batch_
    size]
99.             return in_, out
100.
101.        def on_epoch_end(self):
102.            rng_state = np.random.get_state()
103.            np.random.shuffle(self.x)
104.            np.random.set_state(rng_state)
105.            np.random.shuffle(self.y)
106.
107.    def convconv(filters,size,strides,input):
108.        x = Conv1D(filters,size,strides = strides,padding='same',data_fo
    rmat='channels_first')(input)
109.        x = BatchNormalization()(x)
110.        return x
111.
112.    def convblock1(filters,size,strides,input):
113.        x = convconv(filters,size,strides,input)
114.        x = MaxPooling1D(pool_size=2,padding='same',data_format='channel
    s_first')(x)
115.        return x
```

```
116.
117.  def Build_Model1(b):
118.      inputlayer = Input(shape = (22,250),dtype = np.float32)
119.      X = LSTM(50, return_sequences = False)(inputlayer)
120.      X = Dense(1024,activation='relu')(X)
121.      if b == 1:
122.          Y = Dense(2,activation='softmax')(X)
123.      else:
124.          Y = Dense(5,activation='softmax')(X)
125.      model = Model(inputs = inputlayer, outputs = Y)
126.      optimizer = optimizers.Adam()
127.      if b == 1:
128.          model.compile(optimizer,'binary_crossentropy',['accuracy'])

129.      else:
130.          model.compile(optimizer,'categorical_crossentropy',['accurac
    y'])
131.      return model
132.
133.  def Build_Model2(b):
134.      inputlayer = Input(shape=(22, 250),dtype = np.float32)
135.      X = convblock1(16,3,1,inputlayer)
136.      X = convblock1(32,3,1,X)
137.      X = convblock1(64,3,1,X)
138.      X = convblock1(128,3,1,X)
139.      X = convblock1(256,3,1,X)
140.      X = convconv(512,3,1,X)
141.      X = Flatten()(X)
142.      X = Dense(1024, activation = 'relu')(X)
143.      if b == 1:
144.          Y = Dense(2,activation='softmax')(X)
```

```
145.       else:
146.           Y = Dense(5,activation='softmax')(X)
147.       model = Model(inputs = inputlayer, outputs = Y)
148.       optimizer = optimizers.Adam()
149.       if b == 1:
150.           model.compile(optimizer,'binary_crossentropy',['accuracy'])
151.       else:
152.           model.compile(optimizer,'categorical_crossentropy',['accurac
   y'])
153.       return model
154.
155.   def Build_Model3(b):
156.       inputlayer = Input(shape=(22, 250),dtype = np.float32)
157.       X = convblock1(16,3,1,inputlayer)
158.       X = convblock1(32,3,1,X)
159.       X = convblock1(64,3,1,X)
160.       X = convblock1(128,3,1,X)
161.       X = convblock1(256,3,1,X)
162.       X = convblock1(512,3,1,X)
163.       X = convblock1(1024,3,1,X)
164.       X = convconv(1024,3,1,X)
165.       X = convconv(1024,3,1,X)
166.       X = Flatten()(X)
167.       X = Dense(1024, activation = 'relu')(X)
168.       X = Dense(1024, activation = 'relu')(X)
169.       if b == 1:
170.           Y = Dense(2,activation='softmax')(X)
171.       else:
172.           Y = Dense(5,activation='softmax')(X)
173.       model = Model(inputs = inputlayer, outputs = Y)
```

```
174.     optimizer = optimizers.Adam()
175.     if b == 1:
176.         model.compile(optimizer,'binary_crossentropy',['accuracy'])
177.     else:
178.         model.compile(optimizer,'categorical_crossentropy',['accurac
    y'])
179.     return model
180. 
181. def train(model,epoch,batch_size,b,name):
182.     infos_train = np.arange(NUM_OBSERVATION[0])
183.     infos_val = np.arange(NUM_OBSERVATION[1])
184.     train_x, train_y = load_all(DIR_SAVE_TRAIN, infos_train,b)
185.     val_x, val_y = load_all(DIR_SAVE_VAL, infos_val,b)
186.     # mu = np.mean(train_x)
187.     # std = np.std(train_x)
188.     # print(mu) # -0.8698222
189.     # print(std) # 196.00111
190.     train_x = (train_x-mu)/std
191.     val_x = (val_x-mu)/std
192.     train_batch = DataGenerator(train_x, train_y,batch_size)
193.     val_batch = DataGenerator(val_x, val_y ,batch_size)
194.     model.fit_generator(
195.         generator = train_batch,
196.         steps_per_epoch = len(train_batch),
197.         epochs = epoch,
198.         validation_data = val_batch,
199.         validation_steps = len(val_batch)
200.     )
201.     model.save_weights(name)
202.     return 0
```

```
203.
204.  def test(model,b,name, time_lapse = 0):
205.      infos_test = np.arange(NUM_OBSERVATION[2])
206.      model.load_weights(name)
207.      if time_lapse == 0:
208.          test_x,test_y = load_all(DIR_SAVE_TEST,infos_test, b)
209.      else:
210.          test_x, test_y, k = load_all(DIR_SAVE_TEST,infos_test,b,time
    _lapse)
211.      test_x = (test_x-mu)/std
212.      start = time.time()
213.      y_hat = model.predict(test_x)
214.      end = time.time()
215.      print(test_x.shape[0])
216.      print((end - start)/test_x.shape[0])
217.      accuracy = model.evaluate(test_x,test_y,verbose=2)
218.      print(accuracy)
219.      if time_lapse == 1:
220.          return test_y, y_hat, k
221.      return test_y, y_hat
```

A.1.3 Runner

```
1. import model as ml
2. import os
3. import numpy as np
4. from matplotlib import pyplot as plt
5. from sklearn.metrics import confusion_matrix
6. from sklearn.metrics import roc_curve, auc
7. from itertools import accumulate
8. batch_size = 32
9.
10.   def baseline():
11.       a = np.zeros(6)
12.       for r,d,f in os.walk(ml.DIR_SAVE):
13.           for file in f:
14.               if "l.npy" in file:
15.                   y = np.load(os.path.join(r,file))
16.                   a+=np.sum(y,axis = 0)
17.       return a
18.
19.   def deep_learning(model,name, b, epoch = 100):
20.       np.random.seed(31415)
21.       model.summary()
22.       ml.train(model,epoch,batch_size,b,name)
23.
24.   def ROC_single(y, y_hat):
25.       labels = ['artifact','null']
26.       tpr = dict()
27.       fpr = dict()
28.       roc_auc = dict()
29.       fpr[0], tpr[0], _ = roc_curve(y[:, 0], y_hat[:, 0])
```

```
30.         roc_auc[0] = auc(fpr[0], tpr[0])
31.         plt.figure()
32.         lw = 2
33.         plt.plot(fpr[0], tpr[0], color='darkorange', lw=lw,label='AUC =
    %0.2f' % roc_auc[0])
34.         plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
35.         plt.xlim([0.0, 1.0])
36.         plt.ylim([0.0, 1.05])
37.         plt.xlabel('False Positive Rate')
38.         plt.ylabel('True Positive Rate')
39.         plt.legend(loc="lower right")
40.         plt.show()
41.
42.     def evaluate(model, b, name):
43.         model.summary()
44.         y, y_hat = ml.test(model,b, name)
45.         if b == 0:
46.             cm = confusion_matrix(y.argmax(axis=1),y_hat.argmax(axis=1))
47.             labels = ['eyem','chew','elpp','musc','null']
48.             cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
49.             fmt = '.2f'
50.             fig, ax = plt.subplots()
51.             im = ax.imshow(cm, interpolation='nearest', cmap=plt.cm.Blue
    s)
52.             ax.figure.colorbar(im, ax=ax)
53.             ax.set(xticks=np.arange(cm.shape[1]),
54.                 yticks=np.arange(cm.shape[0]),
55.                 xticklabels=labels, yticklabels=labels,
56.                 ylabel='True label',
57.                 xlabel='Predicted label')
```

```
58.             plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
59.                      rotation_mode="anchor")
60.             thresh = cm.max() / 2.
61.             for i in range(cm.shape[0]):
62.                 for j in range(cm.shape[1]):
63.                     ax.text(j, i, format(cm[i, j], fmt),
64.                         ha="center", va="center",
65.                         color="white" if cm[i, j] > thresh else "black")

66.             fig.tight_layout()
67.             plt.show()
68.
69.         if b == 1:
70.             ROC_single(y,y_hat)
71.
72.     def evaluate_multiple(model1, model2, model3, name1, name2, name3, f
    lag = 0, ensemble = 0):
73.         if flag == 0:
74.             y1, y_hat1 = ml.test(model1,1,name1)
75.             y2, y_hat2 = ml.test(model2,1,name2)
76.             y3, y_hat3 = ml.test(model3,1,name3)
77.         else:
78.             y1 = model1
79.             y_hat1 = model2
80.             y2 = model3
81.             y_hat2 = name1
82.             y3 = name2
83.             y_hat3 = name3
84.         labels = ['artifact','null']
85.         plt.figure()
86.         Y_hat = [y_hat1,y_hat2,y_hat3]
```

```
87.         Y = [y1,y2,y3]
88.         if ensemble == 1:
89.             y4 = (y1+y2+y3)/3
90.             y_hat4 = (y_hat1+y_hat2+y_hat3)/3
91.             Y_hat = [y_hat1,y_hat2,y_hat3,y_hat4]
92.             Y = [y1,y2,y3,y4]
93.
94.         for ii in range(len(Y)):
95.             y_hat = Y_hat[ii]
96.             yy = Y[ii]
97.             if ii == 0:
98.                 label2 = 'RNN '
99.             elif ii == 1:
100.                label2 = 'CNN '
101.            elif ii == 2:
102.                label2 = 'Deep CNN '
103.            else:
104.                label2 = 'Ensemble '
105.
106.            fpr = dict()
107.            tpr = dict()
108.            roc_auc = dict()
109.            for i in range(2):
110.                fpr[i], tpr[i], _ = roc_curve(yy[:, i], y_hat[:, i])
111.                roc_auc[i] = auc(fpr[i], tpr[i])
112.            lw = 2
113.            plt.plot(fpr[0], tpr[0],lw=lw, label = label2+'AUC = %0.2f'
    % roc_auc[0])
114.        plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
115.        plt.xlim([0.0, 1.0])
116.        plt.ylim([0.0, 1.05])
```

```
117.      plt.xlabel('False Positive Rate')
118.      plt.ylabel('True Positive Rate')
119.      plt.legend(loc='lower right')
120.      plt.show()
121.
122.  def evaluate_time_lapse(model, name, window, show = 1):
123.      b = 1
124.      y, y_hat, k = ml.test(model, b , name , time_lapse = 1)
125.      k = list(accumulate(k))
126.      start = 0
127.      y_new = []
128.      y_hat_new = []
129.      for i in range(len(k)):
130.          temp_y = y[start:k[i]]
131.          temp_y_hat = y_hat[start:k[i]]
132.          start = k[i]
133.          for ii in range(temp_y.shape[0]-window+1):
134.              temp = np.sum(temp_y[ii:ii+window],axis=0)
135.              temp = temp[0]>0
136.              if temp:
137.                  y_new.append(np.array([1,0]))
138.              else:
139.                  y_new.append(np.array([0,1]))
140.              y_hat_new.append(np.sum(temp_y_hat[ii:ii+window],axis=0)
    )
141.
142.      y_new = np.array(y_new)
143.      y_hat_new = np.array(y_hat_new)
144.      if show == 1:
145.          ROC_single(y_new,y_hat_new)
146.      return y_new,y_hat_new
```

```
147.
148.  if __name__ == '__main__':
149.      print(baseline()) # Checks baseline statistics
150.      epoch = 100
151.
152.      ######### Multi Class Classification ##############
153.      b = 0
154.      model1 = ml.Build_Model1(b)
155.      name1 = 'RNN_100.h5'
156.
157.      model2 = ml.Build_Model2(b)
158.      name2 = 'CNN_100.h5'
159.      epoch = 30
160.
161.      model3 = ml.Build_Model3(b)
162.      name3 = 'DCNN_100.h5'
163.
164.      ######## Binary Classification ############
165.      b = 1
166.      model1 = ml.Build_Model1(b)
167.      name1 = 'RNN_100_b.h5'
168.
169.      model2 = ml.Build_Model2(b)
170.      name2 = 'CNN_100_b.h5'
171.      epoch = 30
172.
173.      model3 = ml.Build_Model3(b)
174.      name3 = 'DCNN_100_b.h5'
175.
176.      ################### Evaluations #############
177.
```

```
178.      deep_learning(model,name,b, epoch)
179.
180.      evaluate(model1,b,name2)
181.      evaluate(model2,b,name2)
182.      evaluate(model3,b,name3)
183.
184.      evaluate_multiple(model1,model2,model3,name1,name2,name3,ensembl
   e = 1)
185.
186.      y1, y_1 = evaluate_time_lapse(model1,name1,2,show=0)
187.      y2, y_2 = evaluate_time_lapse(model2,name2,2,show=0)
188.      y3, y_3 = evaluate_time_lapse(model3,name3,2,show=0)
189.
190.      evaluate_multiple(y1,y_1,y2,y_2,y3,y_3,flag=1, ensemble = 0)
```