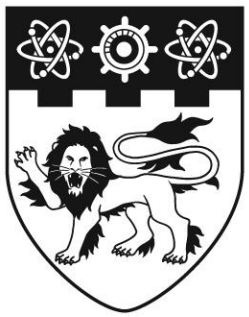

NANYANG TECHNOLOGICAL UNIVERSITY

EEG-Based Emotion Recognition Using Autoencoder and LSTM

Submitted by: Xie Yuan
Matriculation Number: U1722652F

Supervisor: Wang Lipo
Co-supervisor: Olga Sourina

School of Electrical & Electronic Engineering

A final year project report presented to the Nanyang Technological University
in partial fulfillment of the requirements of the degree of
Bachelor of Engineering

2021

Table of Contents

Abstract

EEG-based emotion recognition is one of the key technologies to improving the interaction between computers and the human brain, which can be applied in many industries such as healthcare, entertainment, and psychology. Compared to traditional emotion recognition features (facial, auditory), electroencephalogram (EEG) is considered to be more accurate and more reliable. Thus, in past few years, EEG-based subject-dependent emotion recognition has been intensively investigated using machine learning models such as SVM and KNN. However, many experimental results show that deep learning methods tend to be superior when applying to subject-independent emotion classification. In this thesis, we study a novel deep learning model architecture that utilizes autoencoder model structure to decompose original EEG data into several key signal components and power spectral density (PSD) is extracted, then LSTM recurrent neural network is used to capture the temporal relationship of PSD feature sequence. 66.95% and 70.00% accuracy of positive and negative emotion classification can be achieved in valence and arousal dimensions, respectively. A lot of comparison experiments have been done to try to find the optimal model structure and hyperparameters. Furthermore, we utilized the open-source Python package MNE to help us better understand, visualize, and analyze human EEG data.

Acknowledgments

I would like to express my sincere thanks and great gratitude to my supervisor and co-supervisor **Prof. Wang Lipo** and **Dr. Olga Sourina** for their guidance and advice throughout my final year project.

I also would like to thank **Dr. Lan Zirui** and **Dr. Li Fan** for their great guidance on research direction and for spending precious time answering my questions.

XIE YUAN

May 2021

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Background

With the fast development of modern science and advanced technology, human emotion reflects one's mental state and affective reaction towards an event based on subjective experience [1]. Emotion-related expression not only plays a very important role in human-human communication, but also has potential applications in many industries like healthcare, gaming, and psychology. Researchers are dedicated to finding effective approaches to enhance computers' ability to understand human feelings during the interaction with a human operator. Thus, developing an effective emotion recognition algorithm is the key to enable computers to accurately recognize different emotions from different people so that the machine can truly understand the current mental state of the users. Normally, human emotions are expressed from facial expressions or auditory features, this is how humans identify others' emotions [2]. Similarly, cameras and microphones are often deployed to be the eyes and ears of computers to collect image and audio data so that certain features can be extracted to do the emotion classification task [3]. However, these surface features may not be ideal under some circumstances. For example, when customers are watching a movie in a cinema, the low light condition may

cause difficulties for cameras to capture one's facial expression. On the other hand, according to the experiments by the cinema company, they observed that the audience was almost not changing their facial expression even though the content of the movie was able to change their emotions.

In response, more research has been carried out on other features that can more directly detect human affective states, such as physiological signals [4]. Physiological signals are the readings or measurements that are produced by the physiological process of human beings. For example, heart-beat rate (electrocardiogram or ECG/EKG signal), respiratory rate and content (capnogram), skin conductance (electrodermal activity or EDA signal), muscle current (electromyography or EMG signal), brain electrical activity (electroencephalography or EEG signal). Electroencephalogram (EEG) is one of the most commonly used physiological signals to analyze human emotion states among various types of physiological signals. EEG is a recording of an electrical signal that induced by brain activities, which can be measured from the surface of the scalp using dry electrodes in EEG devices.

With the popularity of portable Brain-computer interface (BCI) devices and the desire for an in-depth understanding of human brain activities, using EEG signals to recognize human emotions is becoming one of the most interested research topics. More and more researchers from universities and industries are making efforts to investigate the relationship between human emotional states and EEG signal patterns. In the past few

years, various kinds of features are studied intensively including time domain, frequency domain, and time-frequency domain features. And machine learning methods are normally applied such as support vector machine (SVM) [5], k-nearest neighbor (KNN) [6] to classify different emotions. With appropriate EEG data pre-processing procedure and model hyperparameter tuning tricks, satisfactory results can be achieved on a subject-dependent basis. 'Subject-dependent' means one machine learning model can only work well on one or several subjects due to the different EEG nature of different people. This is one of the most challenging problems of generally predicting others' emotions using an existing trained model (Subject-dependent).

Subject-dependent emotion recognition is considered to be the most ideal situation. Researchers are trying to solve this problem using the power of deep learning, many novel deep learning models and creative algorithms are proposed. However, currently, there is still no algorithm that can realize fully subject-independent emotion recognition using EEG data. As a result, we are motivated to explore the in-depth relationship between many people's EEG data and their emotions using deep learning techniques such as Convolutional Neural Network (CNN) [7], Long Short-Term model (LSTM) [8], Dense Neural Network (DNN) [9], and so on.

In this thesis, past research methods and results will be reviewed, several types of deep learning models will be explained including their structure and main applications, novel deep learning architecture (SAE+LSTM) [10] will be studied and series of experiments will be executed to compare the result of different methods. We are also

trying to understand the nature of EEG signals by using the Python package MNE visualization tool [11]. In the thesis, the DEAP [12] benchmark dataset will be used to do subject-independent binary emotion classification on valence and arousal dimension separately, with a 10-fold cross-validation standard as many previous research papers did.

1.2 Organizations

The organization of this thesis is as follows:

- Chapter 2 will review some of the research papers that are related to EEG-based emotion recognition using machine learning or deep learning techniques including both subject-dependent and subject-independent. Experiments' settings, dataset been used, feature extraction method, classification method, and other information will be summarised in this part.
- Chapter 3 will explain in detail the methodology used in this thesis including EEG visualization using MNE, various deep learning model explanations, feature extraction procedures, classification methods, comparison experiments explanations. During illustration, screenshots and some Python code will be attached to better demonstrate the methodology used.

- Chapter 4 will elaborate on the execution of a series of comparison experiments and their corresponding results. It will also briefly discuss the possible justifications according to our observation of experiments.
- Chapter 5 will conclude the project and provide further recommendations for this deep learning architecture.

Chapter 2

Literature Review

2.1 Study of EEG

2.1.1 EEG Overview

EEG refers to the recording of the brain's spontaneous electrical activity over a period of time, as recorded from multiple electrodes placed on the scalp. It has been discovered for quite a long time. Richard Caton, an English scientist, is credited with discovering the electrical properties of the brain in 1875, by recording electrical activity from the brains of animals using a sensitive galvanometer, noting fluctuations inactivity during sleep and absence of activity following death. Hans Berger, a German psychiatrist, recorded the first human EEGs in 1924. Later, the first clinical EEG laboratories were established in the United States in the 1930s and 40s [13].

Traditionally, EEG is used to detect abnormal brain activity to determine human brain disorders in the healthcare industry. An EEG might help diagnose or treat epilepsy, brain tumors, stroke, sleep disorders and so on.

In conventional scalp EEG, the recording is obtained by placing electrodes on the scalp. Electrode locations and names are specified by the International 10-20 system [14] for most clinical and research applications. The "10" and "20" refer to the fact that the actual distances between adjacent electrodes are either 10% or 20% of the total front-

back or right-left distance of the skull. Figure 1. Shows the approximate position of different electrodes (DEAP dataset case). Typically, the EEG signal is the voltage difference between the current electrode and the reference electrode (one electrode designed before). A typical adult human EEG signal is about 10 μV to 100 μV in amplitude when measured from the scalp.

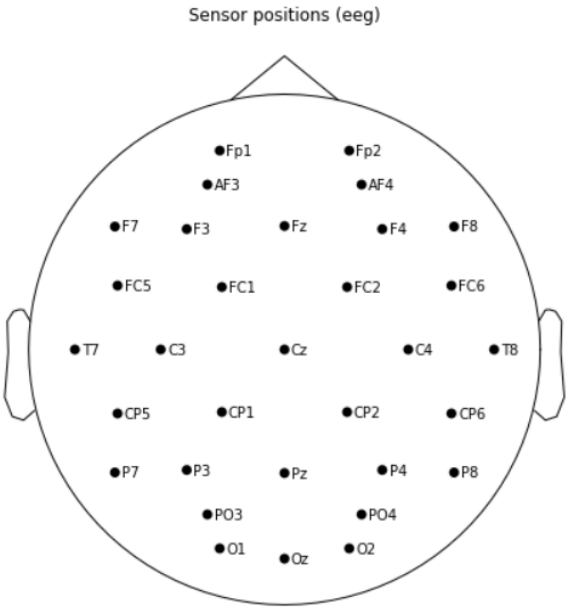

Figure 1. EEG electrodes position in DEAP dataset

2.1.2 EEG features

Frequency bands: EEG signal frequency is an important feature in many use cases. It is normally classified into five bands as shown in Table 1: Delta band, Theta band, Alpha band, Beta band, and Gamma band. The waveforms of these four bands are shown in Figure 2. Every frequency band corresponds to different brain activities. For example, the

Delta wave has a larger amplitude than other bands and typically appears in stages 3 and 4 sleep of humans. But it does not contribute much to the generation of emotions. Instead, human emotion appears to be influenced more by the behavior of Beta and Gamma waves [15].

Band	Frequency (Hz)
Delta wave	< 4 Hz
Theta wave	4-7 Hz
Alpha wave	8-13 Hz
Beta wave	14-30 Hz
Gamma wave	31-50 Hz

Table 1. Five frequency bands

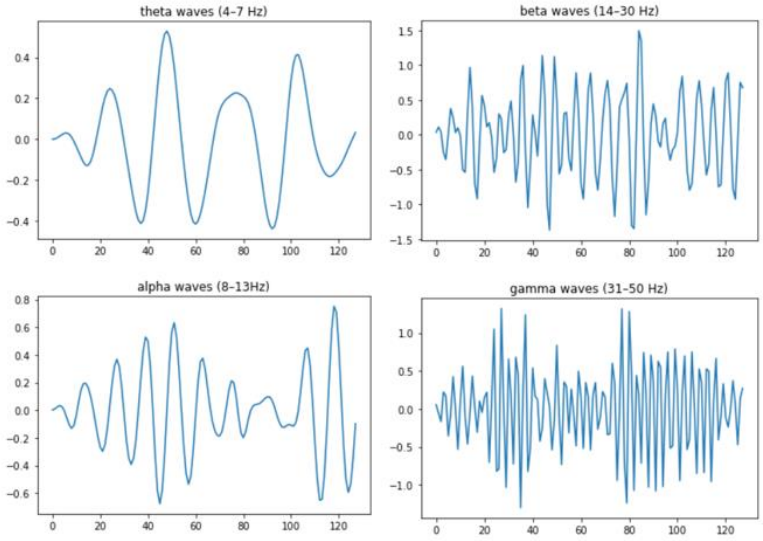

Figure 2. Four band wave using butter bandpass filter

Calculating power spectral density (PSD) [16] of a signal is an effective way to derive frequency band power (FBP) feature that can reveal the composition of different

frequency bands (frequency power distribution) of a fragment of EEG signal. PSD is estimated by calculating the Discrete Fourier transform (DFT) of the signals' auto-correlation function with the formula:

$$F\{x(t) * x(-t)\} = X(f) \cdot X^*(f) = |X(f)|^2,$$

Where $x(t)$ is the time domain signal (EEG signal), F is Fourier transform, $*$ is convolution operation.

Welch's method [17] of PSD calculation reduces noise in the result power spectra by averaging the periodogram of small segments in the original signal, which is widely used in research.

2.2 Continuous 3-dimensional emotion model

To quantitively measure human emotions, state of art 3-dimensional emotion model [2] is widely used in research. Unlike the discrete emotion model, it offers higher resolution when characterizing ambiguous emotions [18]. For example, International Affective Digitized Sounds (IADS) [19] dataset is often used as stimuli of human emotion, which uses valence, arousal, and dominance three orthogonal dimensions to define the components in an emotion. Specifically, for each emotion, the valence dimension represents the pleasure level; the arousal dimension measures the intensity; the

dominance dimension anticipates that how likely the emotion will make humans behave dominantly. Since we will only run our deep learning model on valence and arousal dimensions, from now on we just focus on these two dimensions. As figure 3 shows, two dimensions generate a quadrant diagram, which can result in four general emotions, they are high valence high arousal (HVHA), high valence low arousal (HVLA), low valence high arousal (LVHA), and low valence low arousal (LVLA). Compared to the discrete emotion model, HVHA emotion normally includes alert, excited, happy; HVLA emotion normally includes relaxed, contented, serene; LVHA emotion normally includes upset, nervous, tense; LVLA emotion normally includes sad, depressed, bored.

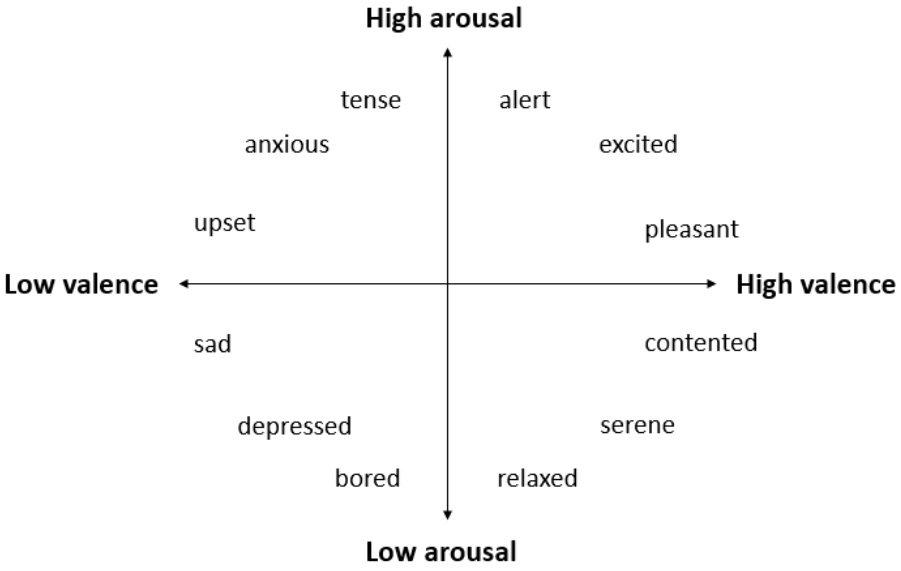

Figure 3. 2-dimensional emotion model

2.3 Review of EEG emotion dataset

2.3.1 DEAP

DEAP emotion dataset [12] will be used in our deep learning model. In this dataset, a total of 32 subjects are experimented, every subject has 40 trials. Each trial is consist of 32 EEG channels, they are: Fp1, AF3, F3, F7, FC5, FC1, C3, T7, CP5, CP1, P3, P7, PO3, O1, OZ, PZ, Fp2, AF4, FZ, F4, F8, FC6, FC2, CZ, C4, T8, CP6, CP2, P4, P8, PO4, O2, the trial length is 63 seconds, 128Hz sampling rate is used, four-dimensional emotions were recorded after watching one music video: valence, liking, arousal, dominance. In our experiments, we consider valence value greater than 5.5 as high valence, valence value less than 4.5 as low valence and so forth.

2.3.2 SEED

SEED [20] is another famous emotion dataset. There are 15 subjects, each of them has 15 trials with 62 EEG channels (AF3, AF4, C1, C2, C3, C4, C5, C6, CB (cerebellum)1, CB2, CP1, CP2, CP3, CP4, CP5, CP6, CPZ, CZ, F1, F2, F3, F4, F5, F6, F7, F8, FC1, FC2, FC3, FC4, FC5, FC6, FCZ, FP1, FP2, FPZ, FT7, FT8, FZ, O1, O2, OZ, P1, P2, P3, P4, P5, P6, P7, P8, PO3, PO4, PO5, PO6, PO7, PO8, POZ, PZ, T7, T8, TP7 and TP8), 200 Hz sampling rate, about 240 seconds. three emotions in discrete emotion model: positive, neutral and negative.

2.4 Review of machine learning method

2.4.1 Support Vector machine

SVM [5] is an effective classification machine learning method that separates different classes by optimizing a hyperplane that maximizes the distances between the hyperplane and these classes. Samples on the boundaries of clusters are called support vectors, where the support vector machine name comes from.

SVM often shows good generalization performance for high dimensional data such as EEG data. [21] proposed methods using universum support vector machine (USVM) and universum twin support vector machine (UTSVM), they achieved an accuracy of 99% when classifying healthy and seizure EEG signals. Wei et al. [22] innovatively applied EEG in SVM model to assess the impact of advertisement, because traditional assessment methods had disadvantages of long cycle times, experimental biases, peer pressure and so on.

SVM with a Radial Basis Function (RBF) kernel is extensively used in EEG emotion recognition because its capability of finding the optimal non-linear hyperplane. Henry et al. [23] used RBF SVM classifier to improve the accuracy of a subject-independent algorithm (binary classification) to 65% by using wavelet features and experimenting with various window sizes ranging from 1-60 seconds, concluding that 3-10 second is the effective window size.

2.5 Review of deep learning method

A summary of related works of the deep learning method is included in Table 2.

2.5.1 Convolutional Neural network

In [24], the authors proposed to use 5-layer CNN to learn EEG features, a 4-layer max pooling to reduce dimensionality, and a fully-connected (FC) layer to do classification. They found that with only 2 electrodes, they can achieve high accuracy in classifying different types of motor imagery electroencephalography (MI-EEG) signals in the Physionet dataset. The electrode pair with the highest accuracy of 98.61% on the 10-subject dataset is FC3-FC4.

EEG signals can also be used to diagnose epilepsy. [25] proposed epileptic EEG signal classification (EESC) method to firstly extract power spectrum density energy diagrams (PSDEDs) features, secondly use CNN to extract features from PSDED and lastly classify the EEG signal into four types of epileptic states. This method could achieve over 90% accuracy in CHB-MIT epileptic EEG data.

Chen et al. [26] combined the EEG feature extracted from the time domain and frequency domain and used a deep convolution neural network to automatically learn other useful features. As a result, proposed EEG features effectively improved both shallow machine learning and deep learning techniques. CVCNN model performed very well with the “FREQNORM” feature, achieving 100% AUC and more than 85% ACC

binary classification both on valence and arousal dimensions but only for within-subject classification.

Recently, [27] proposed a dynamic empirical convolutional neural network (DECNN) that combines the advantages of empirical mode decomposition (EMD) and differential entropy (DE) feature. The first five order IMF components are retained to remove the noise and irrelevant features of the raw signal. Extract DE from each IMF component and each channel contains 5 DE features. Connect the DE features of 29 segments into a one-dimension vector to obtain the time-frequency feature of EEG data and obtain the DDE features of EEG data. After that, the extracted DDE features were classified by a series of convolutional layers. They selected 20 out of 62 channels (FT7, FT8, T7, T8, TP7, TP8, FC5, FC6, C5, C6, F8, F7, F6, F5, FC4, FC3, CP6, CP5, P8, P7), used leave-on-subject-out validation method and achieved 97.56% accuracy on binary classification.

2.5.2 Recurrent Neural network (LSTM)

Long Short-Term Memory (LSTM) is a type of RNN architecture that was designed to model relatively long temporal sequences, which is better than conventional RNNs due to the reduced gradient exploding and gradient vanishing problem [28].

In Google's work [29], LSTM has shown the advantages of processing long sequential data compared to normal RNN. A two-layer deep LSTM achieved state-of-art accuracy of that time in speech recognition applications.

Another type of LSTM, Bidirectional-LSTM, is used to detect seizure from EEG data in [30]. Bi-LSTM is able to make use of temporal information before and after the current timestep. Besides, local mean decomposition (LMD) is used to reduce the complexity of EEG data and statistical feature is extracted. As a result, mean sensitivity achieved 93.61% and a mean specificity achieved 91.85% in seizure detection.

Regarding emotion recognition, Alhagry et al. [31] innovatively proposed an end-to-end deep learning solution that used LSTM to automatically learn features from EEG signals and classified emotions using a dense layer. They achieved 85.65%, 85.45% accuracy in arousal, valence dimensions on the DEAP dataset.

2.5.3 Domain adaptation

Another effective method inspired by computer vision transfer learning techniques is domain adaptation. Domain adaptation techniques have been introduced to help bridge the discrepancy between different subjects.

Lan et al. [2] applied maximum independence domain adaptation (MIDA), Transfer Component Analysis (TCA) and other four domain adaptation techniques to compare the classification result and verify the effectiveness of domain adaptation. It turns out that all

six techniques are able to improve the accuracy by a large amount, especially for MIDA and TCA, they can enhance the mean classification accuracy to up to 72.47% within the SEED dataset.

2.5.4 Autoencoder

An Autoencoder is one kind of deep learning model with symmetric architecture and a bottleneck in the center [10]. It can transform a high-dimensional vector into a latent low-dimensional code (encode process), and then performs a reconstruction from the latent code (decode process) [32].

In classification problems, features are extremely important because they represent the essential nature of the original data. Good features are the key to good classification results. One of the most effective ways to extract features is using autoencoders [33]. It can be used in many areas such as data embedding for visualization, image denoising, anomaly detection and so on.

Regarding EEG-based emotion recognition, research paper [34] showed that autoencoder networks can be used to learn high-order statistical moments information automatically. DE features are extracted and fed into two stacked autoencoders (SAE) and SoftMax classifier, by using five-fold cross-validation and 3 emotions classification, they achieved 85.5% on a subject-dependent basis.

2.5.5 Graph Neural network

Recently, some other researchers are working on finding the relationship between different EEG channels using graph neural networks (GNN). [35] proposed dynamical graph convolutional neural networks (DGCNN) which can effectively extract more discriminative features between multiple channels. They improved the accuracy of subject-dependent experiments and subject-independent experiments up to 90.4% and 79.95% on SEED dataset. Inspired by [35], a similar structure was used in [36], but they considered both local and global inter-channel relationships aimed to find more overlooked features. They also combine it with some transfer learning techniques, specifically, domain adversarial training to better generalize inter-subject experiments. As a result, their algorithm outperformed other competitive baselines and achieved 85.3% accuracy on SEED in subject-independent classification.

2.5.6 Hybrid Neural network

Xin et al. [37] proposed an unsupervised subspace alignment auto-encoder (SAAE) which combined an auto-encoder with a subspace aliment solution to address the limitation of the linear transformation in the existing domain adaptation method and achieved 77.88% mean accuracy on the same SEED dataset. Later, they proposed adaptive subspace feature matching (ASFM) [38] to match both marginal and conditional distributions of source and target domains, they applied a linear transformation function

on marginal distributions which decreased the time complexity of their domain adaptation algorithms while effectively reduced the discrepancies between the source and unlabeled target domain. They improved the accuracy to 80.46% compared to their previous work [37].

Another hybrid deep learning model is proposed in [39]. In this paper, a new EEG feature called multidimensional feature image (MFI) sequences that combines spatial characteristics, frequency domain, and temporal characteristics, are fed to a hybrid deep learning neural network named CLRNN (combine CNN and LSTM to extract spatial features and temporal features). Specifically, the CNN is used to extract features from EEG MFI, and the LSTM is used for modeling the context information of the long-term EEG MFI sequences. CNN has two convolution layers, two max-pooling layers, and a full connection layer. A flatten operation is used and fed into LSTM. At the final layer, softmax gives four probability outputs for four emotions classification. A satisfactory result of 75.21% is gotten on DEAP dataset on a subject-dependent basis.

The main research paper we referred was “SAE+LSTM: A New Framework for Emotion Recognition From Multi-Channel EEG” [10], which uses an autoencoder to decompose the EEG signals to reduce complexity and improves classification accuracy by using the context correlations of the EEG feature sequences using LSTM. PSD from 32 channels EEG data is fed into Stack AutoEncoder (SAE) to extract high-level features and the emotion timing model is based on the Long Short-Term Memory Recurrent Neural Network (LSTM-RNN) with 125 input neurons (since 63-second signal was

divided into 125 segments). Followed by the 125-neuron FC layer, the sigmoid activation function is used in the output layer. For classifier training, the mini-batch gradient descent optimizer and the MSE loss function have been also used. 10-fold cross-validation was used as subject-independent classification, and accuracy of 81.10% in valence and 74.38% in arousal are achieved. In chapter 3, this method will be detailed explained.

Title	**Dataset**	**Method**	**Subject-dependent or subject-independent**	**Accuracy**
Accurate EEG-Based Emotion Recognition on Combined Features Using Deep Convolutional Neural Networks [26]	DEAP	CNN	Subject-dependent (2 classes on valence and arousal)	85.57% on arousal, 88.76% on valence
Subject-independent Emotion Recognition of EEG Signals Based on Dynamic Empirical Convolutional Neural Network [27]	SEED	CNN	Subject-independent (leave-one-subject-out, 2 classes positive and negative)	97.56%
Emotion Recognition based on EEG using LSTM Recurrent Neural Network [31]	DEAP	LSTM	Subject-dependent (four-fold cross-validation, 2 classes on valence, arousal and liking)	85.65%, 85.45% and 87.99% for arousal, valence and liking
EEG-based emotion recognition using machine learning techniques [2]	SEED	Domain adaptation	Subject-dependent (Five-fold cross-validation, 3	72.47%

			Class: positive, neutral, negative)	
EEG-based emotion recognition using machine learning techniques [2]	SEED	Autoencoder	Subject-dependent (Five-fold cross-validation, 3 Class: positive, neutral, negative)	59.66%
Three class emotions recognition based on deep learning using staked autoencoder [34]	SEED	Autoencoder	Subject-dependent (Five-fold cross-validation, 3 Class: positive, neutral, negative)	85.5%
EEG Emotion Recognition Using Dynamical Graph Convolutional Neural Networks [35]	SEED; DREAMER	GNN	Subject-dependent, Subject-independent (leave-one-subject-out) on SEED. Subject-independent (leave-one-session-out) on DREAMER	90.4%, 79.95% on SEED; 86.23%, 84.54% and 85.02% for valence, arousal and dominance on DREAMER
EEG-Based Emotion Recognition Using Regularized Graph Neural Networks [40]	SEED; SEED-IV	GNN	Subject-dependent and Subject-independent both on SEED and SEED-IV	94.24 and 79.37%; 85.30% and 73.84%
Unsupervised domain adaptation techniques based on auto-encoder for non-stationary EEG-based emotion recognition [37]	SEED	Domain adaptation and autoencoder	Subject-dependent (subject-to-subject and session-to-session)	77.88% and 81.81%
Human Emotion Recognition with Electroencephalographic Multidimensional Features	DEAP	CNN and LSTM	Subject-dependent (4 classes: hvha, hvla, lvha, lvla)	75.21%

by Hybrid Deep Neural Networks [39]				
SAE+LSTM: A New Framework for Emotion Recognition From Multi-Channel EEG [10]	DEAP	SAE and LSTM	Subject-independent (10-fold cross-validation, 2 classes on valence and arousal)	81.10% in valence and 74.38% in arousal

Table 2. Summary of EEG-based Emotion Recognition Using Deep Learning

2.6 Review of Python package MNE

MNE is an open-source Python package for exploring, visualizing, and analyzing human neurophysiological data: MEG, EEG, ECoG, NIRS, and more [11]. It has a lot of useful functions like EEG pre-processing, EEG electrode visualization, artifact detection, Independent Components Analysis (ICA) and so on. We will illustrate some of the visualization functions of MNE in this sub-chapter. In our case, we use the DEAP dataset in MATLAB format.

Load one subject data

In [61]:
```
mat = scipy.io.loadmat('DEAP/s01.mat')
data = mat['data'][:, 0:32, :]

# change to mne format # system used in DEAP 'biosemi32'
biosemi32 = mne.channels.make_standard_montage('biosemi32')
info = mne.create_info(ch_names=biosemi32.ch_names, ch_types='eeg', sfreq=128)
raw = mne.EvokedArray(one_data, info) # first trial evoked data

print(data.shape)
print(np.amax(data)) # max value
print(np.amin(data)) # min value
```

```
(40, 32, 8064)
126.13356159993128
-103.35782795738503
```

Figure 4. Code of creating a MNE raw object

As shown in Figure 4, “scipy.io.loadmat()” is used to load Matlab data, and only the first 32 channels are selected. So the data shape of one subject is (40, 32, 8064), representing 40 trials*32 channels*8064 points. “biosemi32” is used as montage since it aligns with DEAP dataset. ‘mne.EvokedArray()’ is to create EvokedArray object that mne uses to do a series of operation.

As shown in Figure 5, “biosemi32.plot()” can be used to plot the topological position of 32 electrodes used in DEAP.

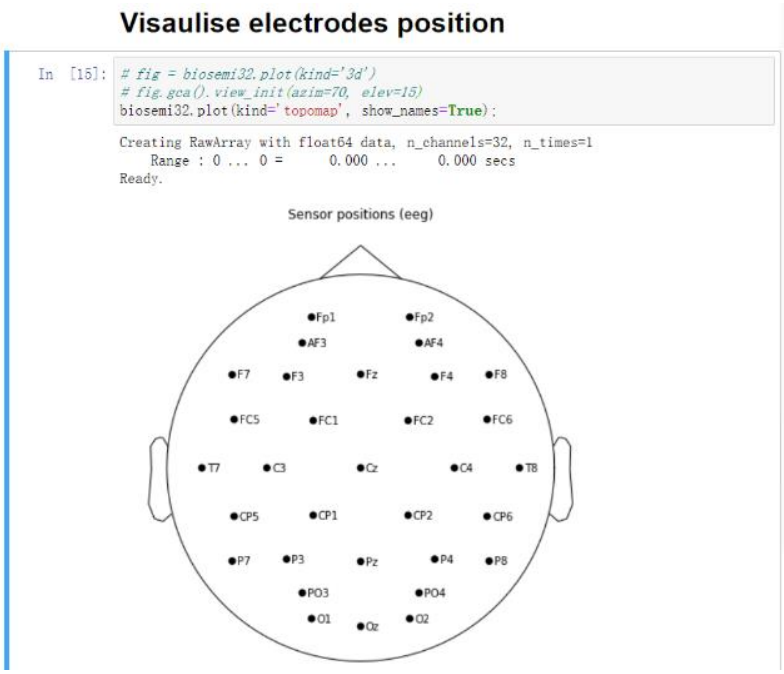

Figure 5. EEG electrodes position in DEAP dataset

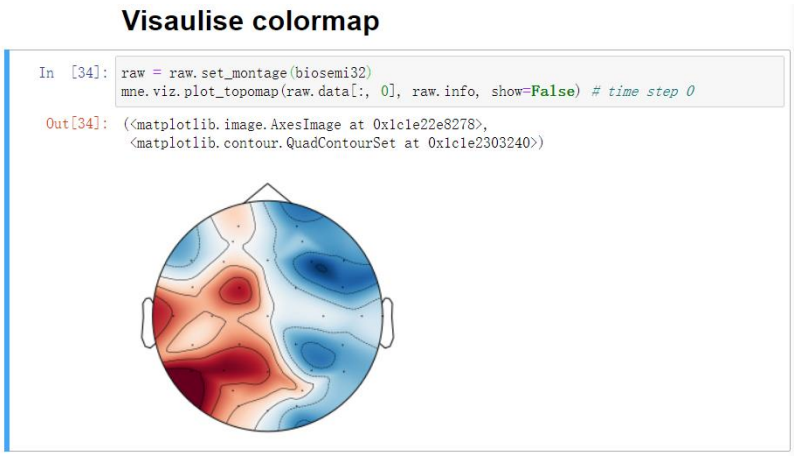

Figure 6. Plot of EEG signal heatmap

As shown in Figure 6, “raw.set_montage()” function is very self-explanatory, after it we can visualize the intensity (high or low in voltage) of EEG signal intuitively.

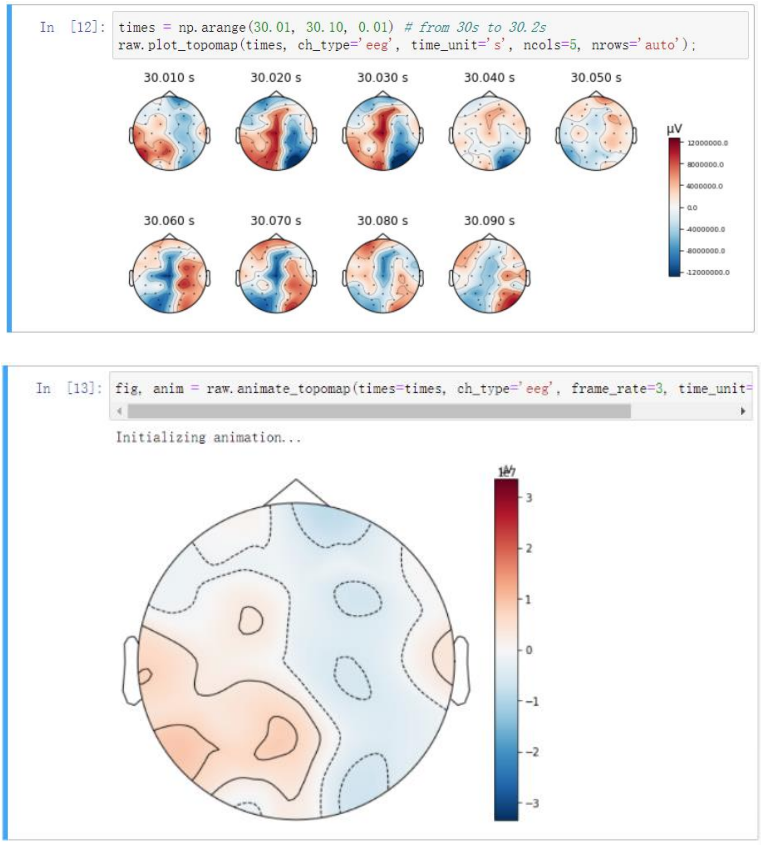

Figure 7. Plot of change of EEG heatmap

As shown in Figure 7, by defining a series of time steps, you can visualize how EEG signal evolves with time using “raw.plot_topomap()” and “raw.animate_topomap()” two functions.

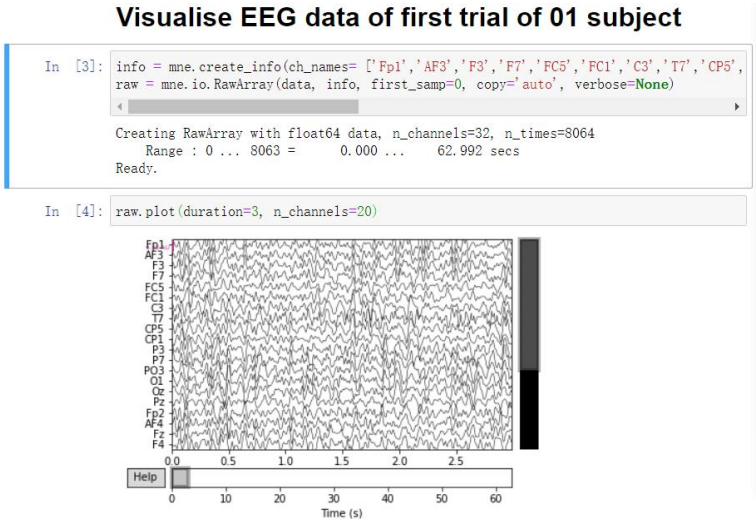

Figure 8. Plot of raw EEG signal

To view how raw EEG looks like, “raw.plot()” can be used, an interactive plot will be displayed to view different channels data as shown in Figure 8.

Chapter 3
Methodology

3.1 Overview

Three main procedures of "SAE+LSTM" [10] will be detailly explained. Firstly, stack autoencoder (SAE) will be used to replace the traditional linear EEG mixing model (as shown in Figure 9). Different from the original paper, it makes use of all 63 seconds of EEG signal, but in our method, we discard the first three seconds signals because they are only in the setup stage. As a result, there are only 119 segments in our method instead of 125 segments. In the original paper, there are 12 different functional brain regions are assumed based on previous research, this linear mixing model can be expressed to:

$$x_1 = a_1 s_1 + a_2 s_2 + \cdots + a_{12} s_{12}$$

Where x1 is the 32-dimensional vector for every time step (every second has 128-time steps because of 128 Hz signal), s_1, s_2, …, s_{12} are the source signal from the 12 different brain regions and a_1, a_2, …, a_{12} are corresponding coefficients.

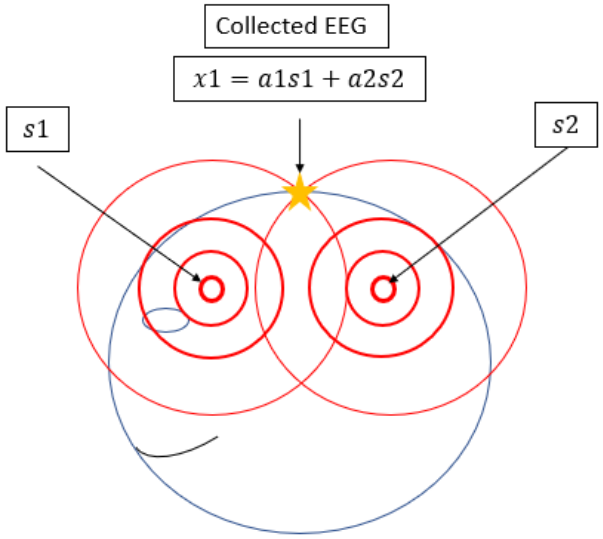

Figure 9. Linear EEG signal mixing model

After training the autoencoder, it can encode the 32 channel EEG signal into 12 channel signal, we assume each channel comes from each brain region. To extract PSD features from each encoded signal, Welch's method is deployed. Then the feature sequence of 119 segments will be fed into an LSTM model with 119 timesteps. Finally, one output will be calculated from each trial, which represents the predicted class (0 or 1) of that trial. The whole process is illustrated in Figure 10.

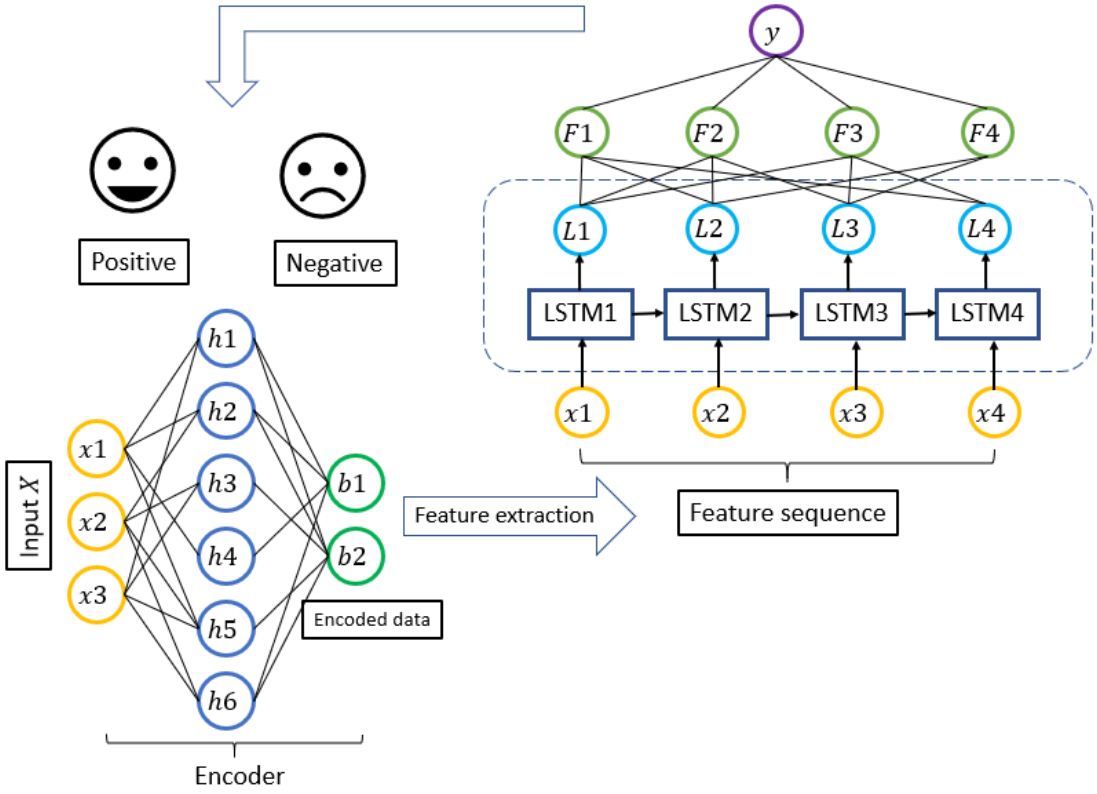

Figure 10. Structure of SAE+LSTM algorithm

3.2 Python packages

As shown in Figure 11, we used numpy, scipy, sklearn keras, mne and other Python packages to implement the algorithm. Also, some constant values are set in this block such as the number of seconds is set to 60 and the number of bottleneck neurons is set to 12.

All Python packages

```
!pip3 install numpy
!pip3 install sklearn
!pip3 install scipy
!pip3 install matplotlib
!pip3 install tensorflow
!pip3 install keras
!pip3 install mne
```

```
import numpy as np
import collections

from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.utils import shuffle

import scipy.io
from scipy import signal, integrate
import matplotlib.pyplot as plt

import keras
from keras.models import Model
from keras.layers import Input, Dense, LSTM, Dropout

import mne
import eeg_entropy
import math

n_second = 60
n_segment = 2*n_second-1
n_points = n_second*128
bottleneck = 12
```

Figure 11. All Python packages needed

3.3 EEG data processing

Deep learning engineers realized that data scaling is a crucial step to conduct because potential gradient exploding or gradient vanishing may severely decrease the accuracy of classification. The two most commonly used effective data scaling techniques are data standardization and data normalization. Data standardization may be especially crucial in order to compare similarities between features based on certain distance measures. In our experiments, we found that data standardization can achieve a better result in autoencoder training.

3.3.1 Data standardization

Typically, data standardization (or Z-score normalization) refers to rescale all data points to have 0 mean and 1 standard deviation, which can be expressed as:

$$z = \frac{x - \mu}{\delta}$$

Where μ is the mean (average) of all data and δ is the standard deviation of all data from the mean value.

StandardScaler() and (a-mean)/std

```
# build in
a = np.array([[1,2,3],[4,5,6],[7,8,9]])
scaler = StandardScaler().fit(a)
a = scaler.transform(a)
print(a)

# custom
a = np.array([[1,2,3],[4,5,6],[7,8,9]])
std = np.std(a)
mean = np.mean(a)
a = (a-mean)/std
print(a)

def standardise_2D(a, multiple):
    std = np.std(a)
    mean = np.mean(a)
    a = (a-mean)/std
    return multiple*a

a = np.array([[1,2,3],[4,5,6],[7,8,9]])
standardise_2D(a, 10)
```

```
[[-1.22474487 -1.22474487 -1.22474487]
 [ 0.          0.          0.        ]
 [ 1.22474487  1.22474487  1.22474487]]
[[-1.54919334 -1.161895   -0.77459667]
 [-0.38729833  0.          0.38729833]
 [ 0.77459667  1.161895    1.54919334]]

array([[-15.49193338, -11.61895004,  -7.74596669],
       [ -3.87298335,   0.        ,   3.87298335],
       [  7.74596669,  11.61895004,  15.49193338]])
```

Figure 12. Code of custom standardization function

To use the built-in function, we can use “from sklearn.preprocessing import StandardScaler, MinMaxScaler” to import two built-in functions. But as shown in Figure 12, the default StandardScaler can only scale data on every column but not the whole matrix. So we need to first find the mean and standardbred derivation and apply the formula above.

3.3.2 Data normalization

Data normalization refers to rescale all data points into a range of 0 to 1 and remain their relative positions (also called unit normalization), but they can be normalized into any range depending on the task. It can be expressed as:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Where x_{norm} is normalized data, x_{min} is the minimum value of all data and x_{max} is the maximum value of all data. For example, in image classification tasks, image pixel values are normally normalized to 0 to 1, which means 0 is the darkest and 1 is the brightest.

MinMaxScaler() and (a - min_) / (max_ - min_)

```
# build in
a = np.array([[1,2,3],[4,5,6],[7,8,9]])
scaler = MinMaxScaler().fit(a)
a = scaler.transform(a)
print(a)

#custom
a = np.array([[1,2,3],[4,5,6],[7,8,9]])
max_, min_ = np.amax(a), np.amin(a)
a = (a - min_) / (max_ - min_)
print(a)

def normalise_2D(a, multiple):
    max_, min_ = np.amax(a), np.amin(a)
    a = (a - min_) / (max_ - min_)
    return multiple*a

a = np.array([[1,2,3],[4,5,6],[7,8,9]])
normalise_2D(a, 1) # range -10 to 10
```

```
[[0.   0.   0. ]
 [0.5 0.5 0.5]
 [1.   1.   1. ]]
[[0.    0.125 0.25 ]
 [0.375 0.5   0.625]
 [0.75  0.875 1.   ]]

array([[0.   , 0.125, 0.25 ],
       [0.375, 0.5  , 0.625],
       [0.75 , 0.875, 1.   ]])
```

Figure 13. Code of a custom normalization function

But as shown in Figure 13, the default MinMaxScaler can only scale data on every column but not the whole matrix. So we need to firstly find the max and min and apply the formula above.

3.4 DEAP EEG data basic process

Convert DEAP dataset from matlab to numpy

```
def convertOneData(file_name):
    mat = scipy.io.loadmat(file_name)
    labels = mat['labels'][:, 0:2] # only valence, arousal, no dominance, liking
    data = mat['data'][:, 0:32, 3*128:] # only first 32 channels ['Fp1','AF3','F3','F7
    #'FC1','C3','T7','CP5','CP1','P3','P7','PO3','O1','Oz','Pz','Fp2','AF4','Fz','F4',
    #'F8','FC6','FC2','Cz','C4','T8','CP6','CP2','P4','P8','PO4','O2']
    #and skip first 3 seconds
    #print(labels.shape, data.shape) # (40, 2) (40, 32, 8064)
    valence_labels, valence_data = [], []
    arousal_labels, arousal_data = [], []
    for i, label in enumerate(labels):
        valence, arousal = label[0], label[1]
        if valence > 5.5: # value >5.5 is high
            valence_labels.append(1)
            valence_data.append(data[i])
        if valence < 4.5: # value <4.5 is high
            valence_labels.append(0)
            valence_data.append(data[i])
        if arousal > 5.5:
            arousal_labels.append(1)
            arousal_data.append(data[i])
        if arousal < 4.5:
            arousal_labels.append(0)
            arousal_data.append(data[i])

    print("valence: ", len(valence_labels), "arousal: ", len(arousal_labels))
    return valence_labels, valence_data, arousal_labels, arousal_data
```

Figure 14. Code of read EEG data and generate high/low labels

As shown in Figure 14, this part of the code is to load one subject's EEG data from MATLAB data file and filter out unnecessary signal information as well as generate the corresponding label for each trial. "scipy.io.loadmat()" is used to read data in MATLAB data file, then iterate through all trials, if the value of valence dimension is greater than 5.5 or less than 4.5, then we consider this trial belongs to "high valence" or "low valence" category, which also applies to arousal dimension. Lastly, the number of valence and arousal will be printed out while reading the data.

```
def convertAllData():
    all_valence_labels, all_valence_data = [], []
    all_arousal_labels, all_arousal_data = [], []
    for i in range(32):
        if i < 10: # subject 01-09
            name = '%0*d' % (2,i+1)
        else: # subject 10-32
            name = i+1
        file_name = "../Data/DEAP/s"+str(name)+".mat"
        print(file_name)
        valence_labels, valence_data, arousal_labels, arousal_data = convertOneData(file_name) # co

        all_valence_labels += valence_labels
        for valence_d in valence_data: # each trial
            valence_d = standardise_2D(valence_d, 1)
            all_valence_data.append(valence_d)
        all_arousal_labels += arousal_labels
        for arousal_d in arousal_data:
            arousal_d = standardise_2D(arousal_d, 1)
            all_arousal_data.append(arousal_d)

    all_valence_labels = np.array(all_valence_labels)
    all_valence_data = np.array(all_valence_data)
    all_arousal_labels = np.array(all_arousal_labels)
    all_arousal_data = np.array(all_arousal_data)
    print("Valence trial data for all subject: ", all_valence_labels.shape,all_valence_data.shape)
    print("Arousal trial data for all subject: ", all_arousal_labels.shape,all_arousal_data.shape)
    # save numpy array of total data to files
    np.save('../Data/processed_DEAP/valence/' + 'all_valence_labels.npy', all_valence_labels)
    np.save('../Data/processed_DEAP/valence/' + 'all_valence_data.npy', all_valence_data)
    np.save('../Data/processed_DEAP/arousal/' + 'all_arousal_labels.npy', all_arousal_labels)
    np.save('../Data/processed_DEAP/arousal/' + 'all_arousal_data.npy', all_arousal_data)
```

Figure 15. Code of iterating through all subjects

The above part of the code in Figure 15 is to read all 32 subject's EEG data by iterating through 32 files and calling "convertOneData" over and over again. At the same time,

data standardization operation is done on every trial by calling “standardize_2D” function. Finally, all valence and arousal EEG data and corresponding labels are saved into “.npy” files. The output of the process is shown in Figure 16:

```
convertAllData()

DEAP/s01.mat
valence:  38 arousal:  39
DEAP/s02.mat
valence:  31 arousal:  34
DEAP/s03.mat
valence:  27 arousal:  36
DEAP/s04.mat
valence:  37 arousal:  33
DEAP/s05.mat
valence:  34 arousal:  31
DEAP/s06.mat
valence:  33 arousal:  33
DEAP/s07.mat
valence:  35 arousal:  37
DEAP/s08.mat
valence:  29 arousal:  27
DEAP/s09.mat
valence:  30 arousal:  31
DEAP/s10.mat
valence:  34 arousal:  34
DEAP/s11.mat
valence:  32 arousal:  34
DEAP/s12.mat
valence:  34 arousal:  31
DEAP/s13.mat
valence:  33 arousal:  37
DEAP/s14.mat
valence:  40 arousal:  40
DEAP/s15.mat
valence:  35 arousal:  28
DEAP/s16.mat
```

Figure 16. Output of convertAllData() function

3.5 Autoencoder

Autoencoder is a deep learning model that has a symmetric structure see from the center layer, which is called the bottleneck layer. The structure of the autoencoder is shown in Figure 17, it clearly shows that it has 32 input neurons at the input layer because the input data is a 32-dimension vector. And the second layer has 64 neurons and the next is the bottleneck layer which has 12 neurons. The same structure is on the right side. To address

the limitation of the linear mixing model, SAE is proposed to become the EEG signal decomposition method because it has a very similar expression to the linear mixing model.

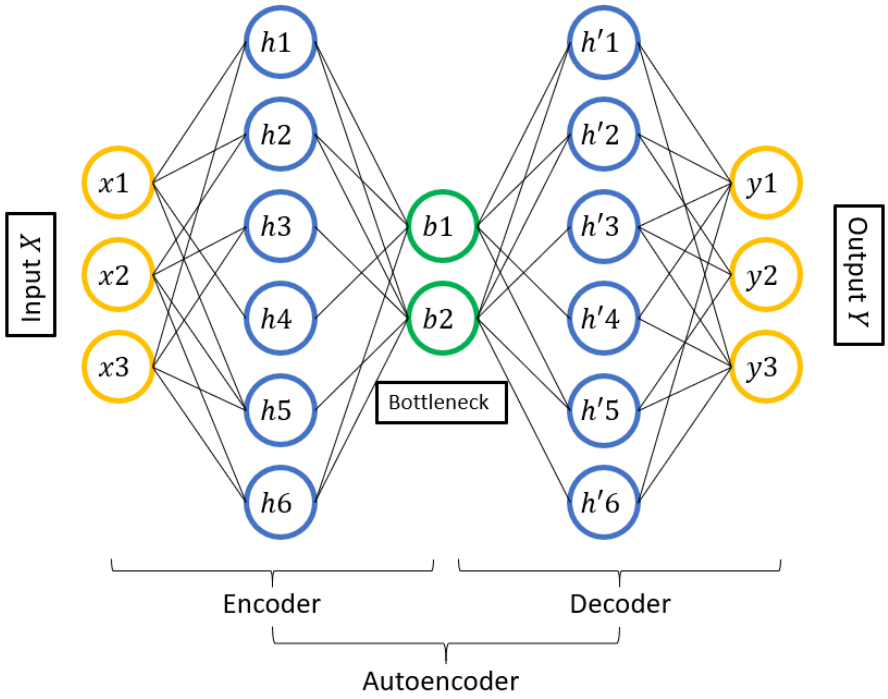

Figure 17. Illustrated stacked autoencoder structure

Figure 18 shows the Python codes that build the structure of our autoencoder, autoencoder has a symmetric structure on the left (encoder) and right (decoder) side, the linear activation function is used in each layer because autoencoder is to replace the linear mixing model. It has a total of 5804 parameters that need to be trained.

Autoencoder structure

```
In [27]: # create new autoencoder
input_layer = Input(shape=(32,))
encoded = Dense(64, activation=None)(input_layer)
bottleneck_layer = Dense(12, activation=None)(encoded)
decoded = Dense(64, activation=None)(bottleneck_layer)
decoded = Dense(32, activation=None)(decoded)
autoencoder = Model(input_layer, decoded)
#autoencoder.summary()
encoder = Model(input_layer, bottleneck_layer)
#encoder.summary()
decoder_input_layer = Input(shape=(12,))
decoder_layer = autoencoder.layers[-2](decoder_input_layer)
decoder_layer = autoencoder.layers[-1](decoder_layer)
decoder = Model(decoder_input_layer, decoder_layer)
#decoder.summary()
```

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 32)]              0
_________________________________________________________________
dense (Dense)                (None, 64)                2112
_________________________________________________________________
dense_1 (Dense)              (None, 12)                780
_________________________________________________________________
dense_2 (Dense)              (None, 64)                832
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080
=================================================================
Total params: 5,804
Trainable params: 5,804
Non-trainable params: 0
```

Figure 18. Code of constructing autoencoder

After constructing the autoencoder model, the format of EEG data has been transformed to fulfill the required format of input data. As shown below, the shape of input data of "vector_transform" function is (x, 32, 7680), x is the number of trials. Firstly, "np.moveaxis" is used to transform the shape into (x, 7680, 32), then, "reshape" is used to reshape the data shape into (x*7680, 32). X*7680 represents that the total number of 32-dimension vectors and every 32-dimension vector is every time step of 32 channels of EEG data. While "inverse_vector_transform" is basically doing the inverse operation of "vector_transform" so that after encoding the training data, it can be used to transform encoded data and extract PSD features of source signals. These two functions are shown in Figure 19.

```
# change dimension from (849, 32, 8064)to (849, 8064, 32) then to (6846336, 32) for input 32-dimension vector to autoencoder
def vector_transform(valence_data):
    valence_vectors = np.moveaxis(valence_data, 1, -1)
    valence_vectors = valence_vectors.reshape((valence_vectors.shape[0]*valence_vectors.shape[1], valence_vectors.shape[2]))
    return valence_vectors

# change output of autoencoder dimension from (6846336, 12) to (849, 8064, 12) then to (849, 12, 8064)
def inverse_vector_transform(valence_vectors):
    valence_data = valence_vectors.reshape((int(valence_vectors.shape[0]/n_points), n_points, valence_vectors.shape[1]))
    valence_data = np.moveaxis(valence_data, -1, 1)
    return valence_data
```

Figure 19. Code of transforming EEG data shape

Stochastic gradient descent (SGD) is used in the autoencoder as an optimizer and Mean Square Error is used as the loss function. SGD is an iterative method for optimizing an objective function with suitable smoothness properties.

In an autoencoder, training data and corresponding labels are the same because the autoencoder is to reconstruct encoded data to be the same as the input data. As a result, the encoded data in the bottleneck layer is the essential element of the original signal.

```
# -------- Compile and train autoencoder --------
autoencoder.compile(optimizer='SGD', loss='mse', metrics=['accuracy'])
autoencoder.fit(train_vectors, train_vectors, epochs=1, batch_size=64, shuffle=True, validation_data=(test_vectors, test_vectors))
autoencoder.save("autoencoder_model/autoencoder_model_test_fold_" + str(test_fold_number))
```

```
96120/96120 [==============================] - 86s 884us/step - loss: 0.2556 - accuracy: 0.4443 - val_loss: 0.2083 - val_accuracy: 0.450
6
INFO:tensorflow:Assets written to: autoencoder_model/autoencoder_model_test_fold_1\assets
```

Figure 20. Code of training autoencoder

As shown in Figure 20, reconstruction accuracy is able to achieve 45.06%. To verify the reconstruction ability of the autoencoder, the original signal and reconstructed signal are plotted using “matplotlib.pyplot”. From the comparison of two images as shown in Figure 21, we can see that the reconstructed signal remains the most characteristic of the original signal.

```
nth = 10
plt.figure(figsize=(15, 3))
plt.plot(data_[0][0][nth*128:nth*128+128]) # first trial first channel
plt.title("Original signal")
plt.xlabel('Points')
plt.ylabel('value')
plt.show()

plt.figure(figsize=(15, 3))
plt.plot(data_decoded[0][0][nth*128:nth*128+128])
plt.title("Reconstructed signal")
plt.xlabel('Points')
plt.ylabel('value')
plt.show()
```

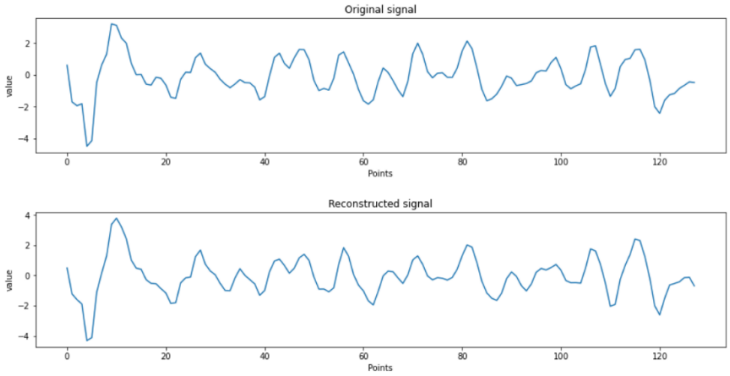

Figure 21. Plot of original signal and reconstructed signal

3.6 Result validation method

In our experiment, we follow the validation method used in the paper, which is 10-fold cross-validation (illustrated in Figure 22). Basically, all EEG trials (1059 trials) are divided into 10 groups randomly, then nine folds of trials are combined as training data (953 trials) and the left one fold is considered as test data (106 trials). So our experiments are repeated 10 times so that every fold has a chance to be the test data, the final result is calculated as the average of 10 experiments.

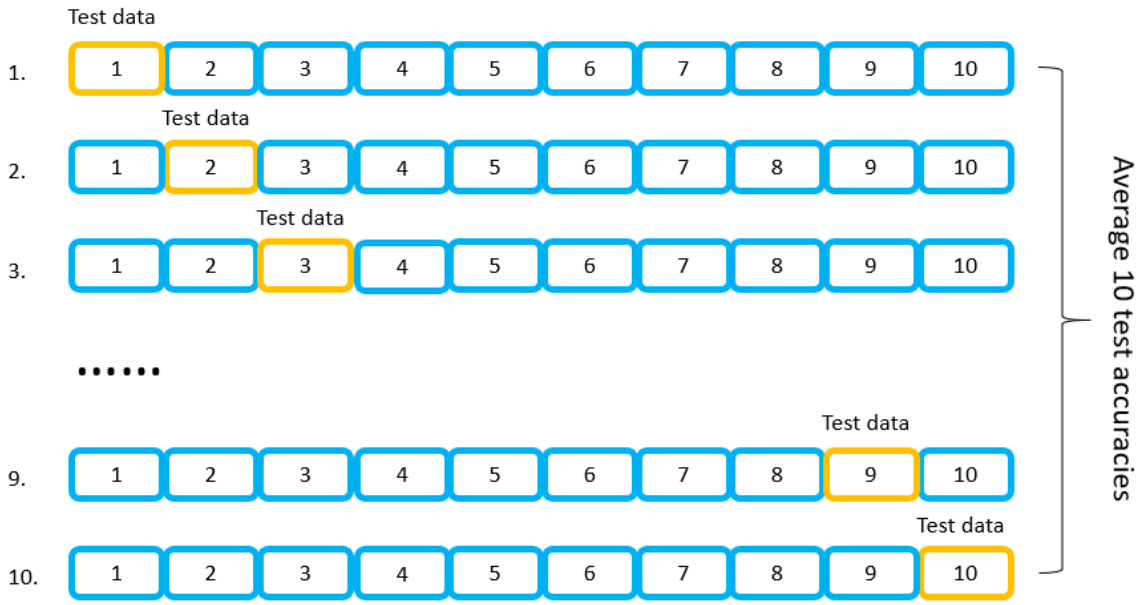

Figure 22. Illustration of 10-fold cross-validation

3.7 Feature extraction method

The main method we use to extract PSD features is the famous Welch's method. Welch's method computes an estimate of the power spectral density by dividing the data into overlapping segments, computing a modified periodogram for each segment and averaging the periodograms. We set the segment length as 1 second and overlap as 0.5 seconds, each segment is multiple with "Hanning window". As shown in Figure 23, the top left is the original 1-second signal (total 128 points), the top right is the Hanning window that generates using "numpy. hanning(128)", and below is the result of multiple of the above two signals.

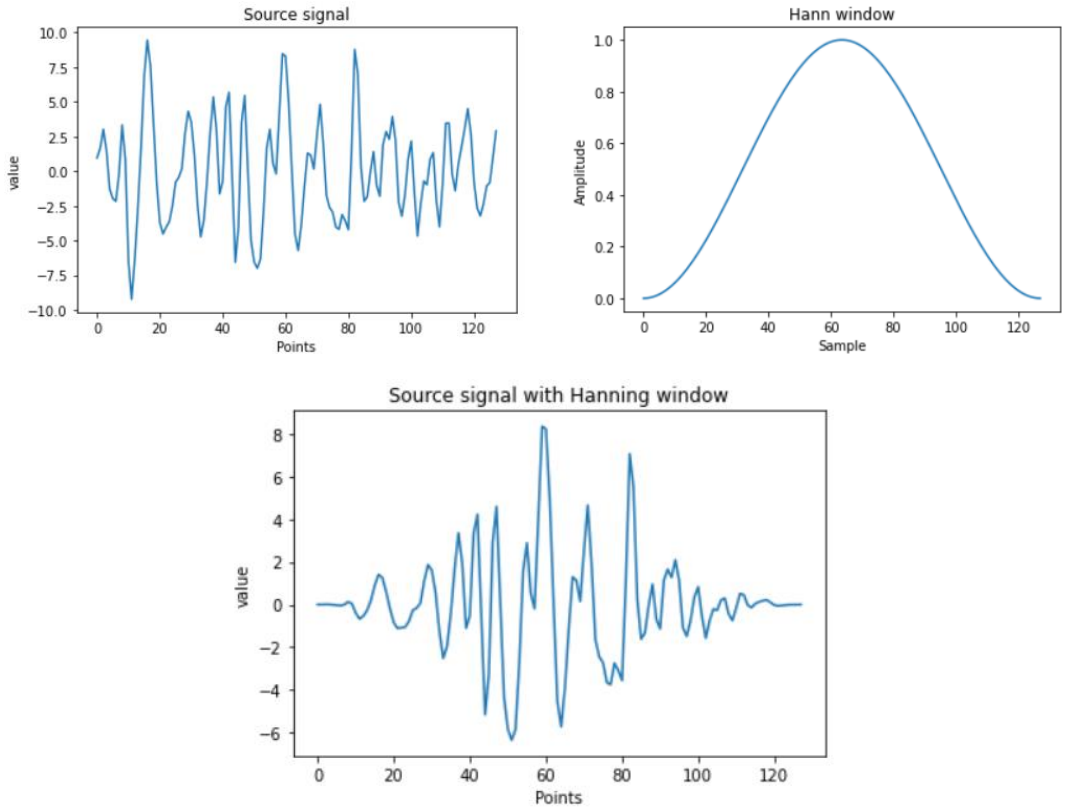

Figure 23. Illustration of Hanning window method

As shown in Figure 24, using “signal.welch()” from scipy can easily calculate the PSD of a signal. Several parameters must specify are EEG signal array, sampling rate, window type used for covering, length of segments and the overlap of segments. In the below Python code, four different frequency bands were colored in the diagram. And FBP features are calculated by integrating the value in each frequency band (the area of each color) so that a four-element array is formed to represent the four FBP features.

```
f, Pxx_den = signal.welch(one_channel_sample, fs=128, window='hann',
                          nperseg = 128, noverlap=64)
plt.plot(f, Pxx_den)
plt.fill_between(f[4:8], Pxx_den[4:8], alpha = 0.3, color='red')
plt.fill_between(f[8:14], Pxx_den[8:14], alpha = 0.3, color='orange')
plt.fill_between(f[14:31], Pxx_den[14:31], alpha = 0.3, color='yellow')
plt.fill_between(f[31:51], Pxx_den[31:51], alpha = 0.3, color='green')
plt.ylim()
plt.xlabel('frequency [Hz]')
plt.ylabel('PSD')
plt.show()
```

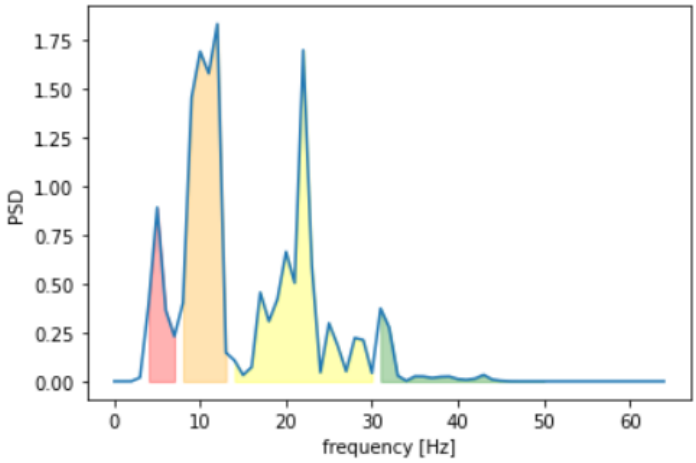

Figure 24. Plot of PSD value according to four bands

As shown in Figure 25, for training data, 953 trails are included, each trail has 119 segments, and each segment has 4 (bands) * 12 (channels) = 48 features.

```
# -------- Feature extraction from 12 source signal --------
train_band_power = [] # band power feature sequence for train trials
for data in train_valence_data_encoded: # for every train trial
    trial_band_power = trial_psd_extraction(data) # data shape (12, 8064)
    train_band_power.append(trial_band_power)
train_band_power = np.array(train_band_power)

test_band_power = [] # band power feature sequence for test trials
for data in test_valence_data_encoded: # for every test trial
    trial_band_power = trial_psd_extraction(data) # data shape (12, 8064)
    test_band_power.append(trial_band_power)
test_band_power = np.array(test_band_power)
print("All features of training data shape: ", train_band_power.shape) # shape (953, 125, 48)
print("All features of test data shape: ", test_band_power.shape) # shape (106, 125, 48)
```

Figure 25. Code of applying feature extraction function

3.8 LSTM-RNN

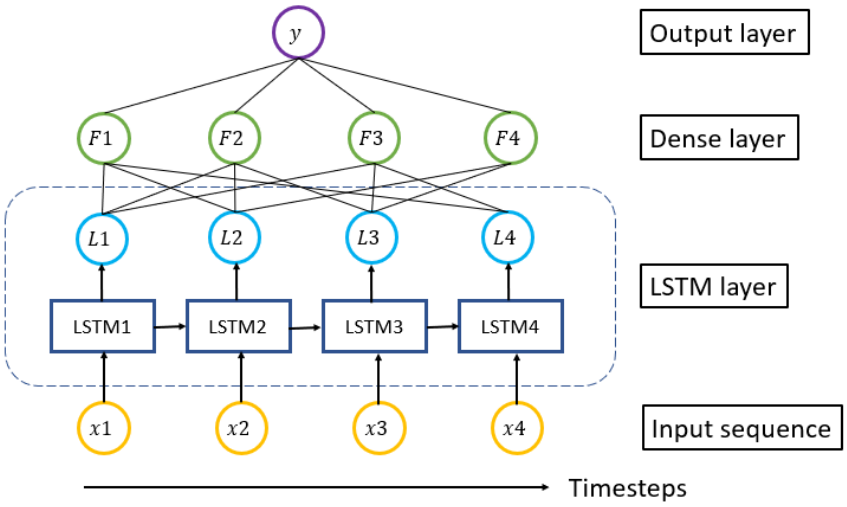

Figure 26. Structure of LSTM and dense layer

Figure 26 illustrates the overall structure of LSTM used in this thesis. Since there are 119 segments in one trial and 48 features are extracted from one segment, the input shape of the LSTM input layer is set to (119, 48). One LSTM layer is followed. The output from the LSTM layer is sent to two dense layers with 119 and 12 neurons. Lastly, a dense layer with a “sigmoid” activation function is used to output final classification accuracy. The same SGD optimizer and MSE are used in LSTM, the training epoch is set to 30, the batch size is 8. The training data is all the feature sequence (FBP) calculated above and labels are the corresponding 0 or 1 label of original trials. Figure 27 shows the code of constructing the LSTM model, Figure 28 shows the training of the LSTM model.

```
# -------- Create new LSTM model --------
x=Input(shape=(n_segment,bottleneck*4)) # flatten (12,4) to 48
x1=LSTM(n_segment)(x)
x2=Dense(n_segment)(x1)
x3=Dense(12)(x2)
output=Dense(1, activation="sigmoid")(x2)
model=Model(x, output)
```

Figure 27. Code of constructing LSTM model

```
# -------- Compile and train LSTM --------
model.compile(optimizer='SGD', loss='mse', metrics=['accuracy'])
history = model.fit(train_band_power, train_labels, epochs=30, batch_size=8, validation_data=(test_band_power, test_labels))
print("Hightest accuracy: " + str(max(history.history['val_accuracy'])))
model.save("../Results/LSTM_model/LSTM_model_test_fold_" + str(test_fold_number))
```

```
for i in range(10):
    print("********** Test Fold " + str(i) + " ************")
    process(i)
```

```
122/122 [==============================] - 7s 60ms/step - loss: 0.2065 - accuracy: 0.6922 - val_loss: 0.2148 - val_accuracy: 0.6759
Epoch 6/30
122/122 [==============================] - 7s 61ms/step - loss: 0.1996 - accuracy: 0.7043 - val_loss: 0.2070 - val_accuracy: 0.7037
Epoch 7/30
122/122 [==============================] - 8s 64ms/step - loss: 0.1928 - accuracy: 0.7230 - val_loss: 0.2075 - val_accuracy: 0.7315
Epoch 8/30
122/122 [==============================] - 8s 62ms/step - loss: 0.1900 - accuracy: 0.7260 - val_loss: 0.2037 - val_accuracy: 0.7315
Epoch 9/30
122/122 [==============================] - 8s 62ms/step - loss: 0.1898 - accuracy: 0.7155 - val_loss: 0.2034 - val_accuracy: 0.7315
Epoch 10/30
122/122 [==============================] - 7s 61ms/step - loss: 0.1827 - accuracy: 0.7287 - val_loss: 0.2042 - val_accuracy: 0.7407
Epoch 11/30
122/122 [==============================] - 8s 62ms/step - loss: 0.1844 - accuracy: 0.7439 - val_loss: 0.2042 - val_accuracy: 0.7222
Epoch 12/30
122/122 [==============================] - 8s 62ms/step - loss: 0.1729 - accuracy: 0.7588 - val_loss: 0.2076 - val_accuracy: 0.6759
Epoch 13/30
122/122 [==============================] - 8s 63ms/step - loss: 0.1867 - accuracy: 0.7292 - val_loss: 0.2000 - val_accuracy: 0.7407
Epoch 14/30
122/122 [==============================] - 8s 63ms/step - loss: 0.1691 - accuracy: 0.7614 - val_loss: 0.1923 - val_accuracy: 0.7500
```

Figure 28. Code of training LSTM model and result

The result of the LSTM model is shown in Figure 28, as we can see, the validation accuracy varies a bit since we only have 106 test data. Thus, we consider the highest accuracy of every experiment (in the first experiment the highest accuracy is 75%) and compute the average accuracy of 10 experiments as our metric to compare with the following experiments.

3.9 Comparison experiments

3.9.1 LSTM without autoencoder

To validate the usefulness of the autoencoder in our method, we have done the comparison experiment which does not make use of autoencoder to decompose the original 32-channel EEG data. The same method is used to extract the PSD feature as stated above. The only difference is that instead of using a 12-channel signal, a 32-channel signal is used. Thus, for train data, 953 trails are included, each trail has 119 segments, and each segment has 4 (bands) * 32 (channels) = 128 features. Accordingly, the input shape of the LSTM input layer becomes (119, 128) as shown in Figure 29.

```
# -------- Feature extraction from 32 original signal --------
train_band_power = [] # band power feature sequence for train trials
for data in train_data: # for every train trial
    with io.capture_output() as captured:
        trial_band_power = trial_psd_extraction_integration(data) # data shape (32, 8064)
    train_band_power.append(trial_band_power)
train_band_power = np.array(train_band_power)

test_band_power = [] # band power feature sequence for test trials
for data in test_data: # for every test trial
    with io.capture_output() as captured:
        trial_band_power = trial_psd_extraction_integration(data) # data shape (32, 8064)
    test_band_power.append(trial_band_power)
test_band_power = np.array(test_band_power)
print("All features of training data shape: ", train_band_power.shape) # shape (849, 125, 128)
print("All features of test data shape: ", test_band_power.shape) # shape (95, 125, 128)
```

Figure 29. Feature extraction result format

With all other settings are the same as the previous experiment, the epoch achieved the highest result of 69.44% accuracy as shown in Figure 30. Other results of the first comparison experiment are shown in chapter 3.10.

```
    # -------- Compile and train LSTM --------
    model.compile(optimizer='SGD', loss='mse', metrics=['accuracy'])
    history = model.fit(train_band_power, train_labels, epochs=30, batch_size=8, validation_data=(test_band_power, test_labels))
    print("Hightest accuracy: " + str(max(history.history['val_accuracy'])))
    model.save("../Results/LSTM_model/LSTM_model_test_fold_" + str(test_fold_number))
```

```
for i in range(10):
    print("********** Test Fold " + str(i) + " ************")
    process(i)
```

```
********** Test Fold 0 ************
(972, 32, 7680) (108, 32, 7680)
All features of training data shape:  (972, 119, 128)
All features of test data shape:  (108, 119, 128)
Epoch 1/30
122/122 [==============================] - 9s 59ms/step - loss: 0.2575 - accuracy: 0.5167 - val_loss: 0.2316 - val_accuracy: 0.6389
Epoch 2/30
122/122 [==============================] - 7s 59ms/step - loss: 0.2220 - accuracy: 0.6501 - val_loss: 0.2239 - val_accuracy: 0.6389
Epoch 3/30
122/122 [==============================] - 7s 58ms/step - loss: 0.2174 - accuracy: 0.6887 - val_loss: 0.2211 - val_accuracy: 0.6481
Epoch 4/30
122/122 [==============================] - 7s 59ms/step - loss: 0.2049 - accuracy: 0.6957 - val_loss: 0.2225 - val_accuracy: 0.6204
Epoch 5/30
122/122 [==============================] - 7s 58ms/step - loss: 0.1883 - accuracy: 0.7549 - val_loss: 0.2197 - val_accuracy: 0.6667
Epoch 6/30
122/122 [==============================] - 7s 55ms/step - loss: 0.1855 - accuracy: 0.7554 - val_loss: 0.2210 - val_accuracy: 0.6944
Epoch 7/30
122/122 [==============================] - 7s 58ms/step - loss: 0.1898 - accuracy: 0.7222 - val_loss: 0.2127 - val_accuracy: 0.6944
```

Figure 30. Result of first comparison LSTM accuracy

3.9.2 SVM method

The second comparison experiment is set up as baseline accuracy, which is using SVM as the classifier. Since LSTM was not used in the experiment, the method of PSD feature extraction also changed. For every channel of EEG data, welch's method was used to compute PSD value, then integration was used to compute the FBP in four different bands. So for each trial, a total of 32 * 4 features were extracted and flattened into a 1-dimension vector according to Figure 31. One example of 128 features is shown in Figure 32:

```python
def trial_psd_extraction2(data): # data shape (12, 7680)
    all_channels_psd = []

    for channel in data:
        f, Pxx_den = signal.welch(channel, fs=128, window='hann', nperseg = 128, noverlap=64)
        y_int = integrate.cumtrapz(Pxx_den, f, initial=0) # integrate to calculate band power
        channel_psd = np.array([y_int[7]-y_int[4],y_int[13]-y_int[8],y_int[30]-y_int[14],y_int[51]-y_int[31]])
        all_channels_psd.append(channel_psd)

    all_channels_psd = np.array(all_channels_psd)
    all_channels_psd = all_channels_psd.reshape((32*4))

    return all_channels_psd
```

Figure 31. Feature extraction method and feature of one trial for SVM

```
trial_psd_extraction2(all_valence_data[0])
```

```
array([0.20647754, 0.27151231, 0.32374936, 0.04267095, 0.2738797 ,
       0.2991804 , 0.36725694, 0.03897056, 0.30901138, 0.32617527,
       0.39162152, 0.04443694, 0.31216755, 0.3566092 , 0.31651962,
       0.05305372, 0.15131731, 0.19556045, 0.16550802, 0.03538698,
       0.1516904 , 0.16776573, 0.20368998, 0.02898709, 0.16636402,
       0.25327529, 0.18517809, 0.03578409, 0.33963168, 0.37669346,
       0.41730986, 0.1768792 , 0.12820309, 0.3664592 , 0.20504746,
       0.04285048, 0.07750988, 0.19091999, 0.1392715 , 0.01736153,
       0.32195878, 0.5264116 , 0.28377465, 0.05714146, 0.22133074,
       0.54883728, 0.33020767, 0.10478794, 0.18993892, 0.6633732 ,
       0.26088971, 0.05927537, 0.23711966, 0.4894831 , 0.32668154,
       0.11765391, 0.25046878, 0.40704044, 0.25176368, 0.05182232,
       0.10678177, 0.28809704, 0.15476428, 0.01824415, 0.37497274,
       0.31760395, 0.33250989, 0.04815584, 0.27597607, 0.29966256,
       0.29409691, 0.05840552, 0.32658178, 0.29978938, 0.31423666,
       0.02838681, 0.39104178, 0.3382205 , 0.34324028, 0.04231173,
       0.21777229, 0.32238494, 0.33381327, 0.05314186, 0.13886796,
       0.25159971, 0.21788824, 0.03575721, 0.27888568, 0.24778188,
       0.25537265, 0.03005991, 0.11376716, 0.1002034 , 0.20680615,
       0.01870352, 0.14222308, 0.22894292, 0.20501456, 0.03248267,
       0.30435638, 0.37662   , 0.31195382, 0.08586265, 0.08481513,
       0.24955586, 0.18028833, 0.03214722, 0.08342442, 0.15373   ,
       0.14147418, 0.01642434, 0.42908997, 0.45472968, 0.33457299,
       0.067645  , 0.14634095, 0.30985876, 0.20313569, 0.0625774 ,
       0.35584632, 0.50435648, 0.29195788, 0.04223875, 0.2786323 ,
       0.45208378, 0.25862115, 0.03399016])
```

Figure 32. One example of 128 features

The creation of the SVM model is relatively easy. As explained before, the “RBF” kernel of the SVM model has advantages that can better distinguish between different classes. Then, the features extracted above were fed into the SVM model, keeping all hyperparameter as default as shown in Figure 33:

```
# -------- Create new SVM model --------
from sklearn import svm
from sklearn import metrics

clf = svm.SVC(kernel='rbf')
clf.fit(train_band_power, train_valence_labels)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
y_pred = clf.predict(test_band_power)
print("Accuracy: ",metrics.accuracy_score(test_valence_labels, y_pred))
```

```
Accuracy:  0.5943396226415094
```

Figure 33. Code of constructing SVM model and validation results

After a very quick training process, 'metric' in sklearn was used to output the validation accuracy based on test data. As we can see, the classification accuracy for the first fold is 59.43%. Other results of the second comparison experiment are shown in 3.10.

3.10 Results

The results of the SAE+LSTM method (10-fold validation) on valence dimension are shown in Table 3, the average result is 66.95%.

Experiment No.	1	2	3	4	5
Accuracy	63.81%	66.67%	61.90%	68.57%	69.52%
Experiment No.	6	7	8	9	10
Accuracy	70.48%	66.67%	66.67%	64.76%	70.48%

Table 3. SAE+LSTM method 10 experiments results on valence dimension

The results of the SAE+LSTM method (10-fold validation) on arousal dimension are shown in Table 3, the average result is 70.00%.

Experiment No.	1	2	3	4	5
Accuracy	75.00%	74.07%	68.51%	64.81%	66.67%
Experiment No.	6	7	8	9	10
Accuracy	72.22%	70.37%	66.67%	67.59%	74.07%

Table 4. SAE+LSTM method 10 experiments results on arousal dimension

The results of the first comparison experiment (LSTM without autoencoder,10-fold validation) on valence dimension are shown in Table 4, the average result is 63.81%.

Experiment No.	1	2	3	4	5
Accuracy	66.67%	68.57%	59.05%	66.67%	60.95%
Experiment No.	6	7	8	9	10
Accuracy	60.95%	59.05%	61.90%	69.52%	64.76%

Table 5. LSTM without SAE 10 experiments results on valence dimension

The results of the first comparison experiment (LSTM without autoencoder,10-fold validation) on arousal dimension are shown in Table 4, the average result is 69.53%.

Experiment No.	1	2	3	4	5
Accuracy	69.44%	68.51%	70.37%	69.44%	72.22%
Experiment No.	6	7	8	9	10
Accuracy	70.37%	69.44%	69.44%	64.81%	71.30%

Table 6. LSTM without SAE 10 experiments results on arousal dimension

The average of 10 experiments accuracy on valence is 63.81%, which is a little bit lower than the SAE+LSTM experiment (66.95%). This result indicated that it is necessary to use some kind of decomposition method. In our case, the deep learning method autoencoder is used, which reduces the complexity of original EEG data and improves the classification accuracy. The average accuracy on arousal is 69.53%, which is almost the same as SAE+LSTM (70.00%).

The results of the second comparison experiment (SVM, 10-fold validation) on valence dimension are shown in Table 5, the average result is 58.57%.

Experiment No.	1	2	3	4	5
Accuracy	57.14%	53.33%	60.95%	60%	53.33%
Experiment No.	6	7	8	9	10
Accuracy	63.81%	54.29%	62.86%	57.14%	62.86%

Table 7. SVM 10 experiments results on valence

The results of the second comparison experiment (SVM, 10-fold validation) on arousal dimension are shown in Table 5, the average result is 65.74%.

Experiment No.	1	2	3	4	5
Accuracy	71.30%	69.44%	63.89%	62.03%	59.26%
Experiment No.	6	7	8	9	10
Accuracy	69.44%	64.81%	69.44%	61.11%	66.67%

Table 8. SVM 10 experiments results on arousal

The average of 10 experiments' accuracy on valence is 58.57%, which is lower than the LSTM without SAE experiment (63.81%) and also inferior to the SAE+LSTM experiment (66.95%), which indicates the usefulness of the LSTM model.

The classification result in the original paper is able to achieve 81.1% on valence and 74.38% on arousal while in our implementation only 66.95% and 70.00% is achieved. It is worth noting that the differences in implementation may lead to quite a different result, especially on valence dimension. Compared to result 59.66% (Subject-dependent, positive, neutral, negative emotions) of [2], which only used autoencoder to extract essential feature; and compared with [31], whose subject-dependent (two classes) is 85.45%, our subject-independent result 66.95% and 70.00% is satisfactory but still needs to be improved.

The comparisons of different experiments results are presented in Figure 34.

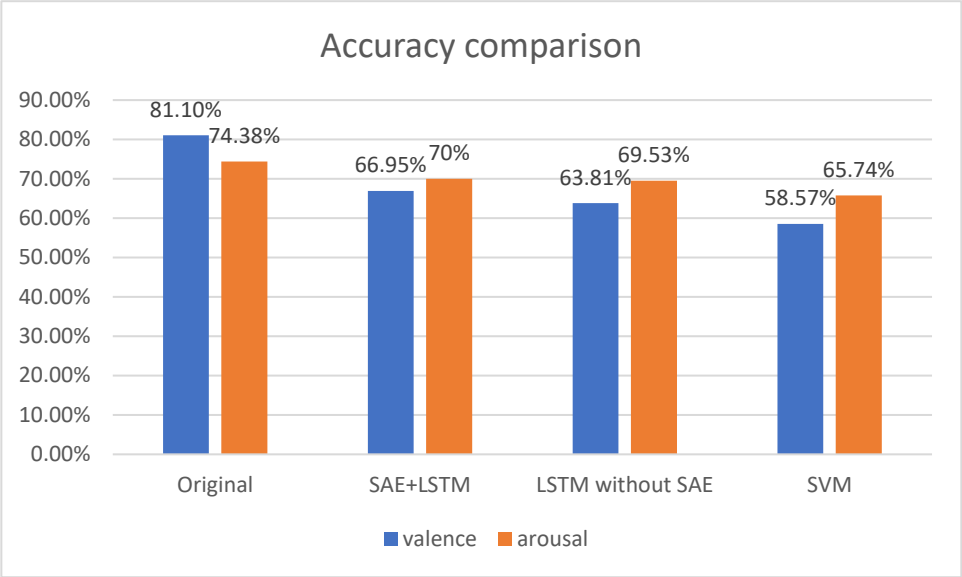

Figure 34. Accuracy comparison

Chapter 4

Conclusions and Future Work

4.1 Conclusions

In this thesis, we have introduced the background of current EEG emotion recognition, studied some of the effective methods using machine learning and deep learning techniques, and conducted a series of experiments following the selected SAE + LSTM method. The average of highest classification accuracy of 10 validation experiments is 66.95% (subject-independent) and 10-fold cross-validation is used. Compare this result with the latter two comparison experiments, we have found that the autoencoder + LSTM method (accuracy 66.95% and 70.00% on valence and arousal) outperforms both comparison experiment 1 (LSTM without autoencoder, accuracy 63.81% and 69.53% on valence and arousal) and experiment 2 (SVM method, accuracy 58.57% and 65.74% on valence and arousal). We have validated that SAE + LSTM with PSD feature methodology do have the advantages of reducing the complexity of the original EEG signal and exploiting the frequency and temporal information of the EEG signal.

4.2 Recommendation in Future Work

Although the autoencoder + LSTM method has proven to have advantages in recognizing human emotion quite accurately, the classification result in the original paper is able to

achieve 81.1% on valence while in our implementation only 66.95% is achieved. So it is worth noting that the differences in implementation may lead to quite a different result. In the original paper, details of EEG data processing techniques are not presented such as data normalization and data standardization. But according to our experiment result, data standardization is preferred. Thus, finding correct and optimal methods of implementation in experiments is the key to achieve high classification results. In future research work, more research papers need to be studied to further understand the mathematical principle behind EEG feature extraction, and more experiments need to be done to verify if the proposed method can achieve the requirements we intended to see. Furth more, new algorithms that combine different features (temporal, spatial, frequency and so on) and different deep learning techniques (CNN, GNN, LSTM and so on) need to be proposed to achieve higher subject-independent emotion recognition.

Reflection on Learning Outcome Attainment

Engineering knowledge

Throughout my Final Year Project, I obtained a lot of engineering knowledge besides the curriculum course. EEG was a new area that I had never learned about so I needed to read several papers to understand the nature of EEG signals and current research achievements about EEG-based emotion recognition. The ph.D. thesis of Lan Zirui [2] really explained the fundamental knowledge and state-of-art techniques of EEG such as SVM, domain adaptation, basic autoencoder, which were very informative for a new researcher. Other more recent papers also proposed many new deep learning methods rather than machine learning methods to improve the accuracy of subject-independent emotion recognition. From these papers, I learned some basics about CNN, RNN, GNN, advanced autoencoder and so on. In this FYP report, I selected SAE + LSTM method [10] as the main method to experiment with. As an undergraduate student, it was not easy to implement a paper with only a few details given. So I had to search online to understand the basic principles and find example code to implement what I need, this really improved my research skills and self-learning skills.

Design/development of Solutions

SAE + LSTM method consists of a series of procedures, so I divided the whole process into several parts, and solve them one by one. The first part is to load EEG data from the .mat files into Python, filter out unnecessary data, standardize EEG data and generate labels for later model training. The second part is to transform data format, construct an autoencoder deep learning model, train autoencoder model and generate source signals from the encoder. The third part is to extract PSD features from source signals according to the expected format. The final part is to construct the LSTM model, feed features into the LSTM model and generates the final classification result. Throughout these parts, many experiments had been done to verify the correctness of the code. Unexpected results were encountered occasionally, so debugging was been done to get the expected result. FYP greatly improved my analytic skills and logical thinking.

Modern Tool Usage

The main tool I used was the Python programming language. Python is a very powerful language that has plenty of packages so that many different kinds of data can be processed. It has the advantages of easy to understand and fast to prototype project. In my project, I have used sikitlearn, scipy, matplotlib, numpy, mne and so on. MNE is a quite new python package related to EEG that provides many functions like EEG preprocessing, EEG visualization and EEG feature extraction. I have used it to visualize

EEG data in different emotions, extract PSD features and so on, it is very well designed and well documented. Another very useful Python package is Keras, which is a high-level API for easy deep learning model construction. I have used it to build autoencoder layer by layer and it is very intuitive to use. By using these powerful tools, I improved my programming skills and ability to explore new useful tools.

Lifelong Learning

There is still a lot of room for the progress of high accurate EEG-based emotion recognition, and more efforts need to be taken to investigate the deeper nature of EEG data. As a result, we should not stop the research on EEG, actually, it just started. As more powerful deep learning methods were proposed, more deep features may be discovered that can better represent different emotions. In the future, with the development of general artificial intelligence, such as the powerful transformer model, we can apply a transformer to classify emotion based on EEG. In summary, more and more research is focusing on EEG analysis, and more applications are proposed that are related to human EEG data, we need to keep learning, keep applying new technology to EEG applications, we may get more surprising results.

References

[1] I. B. Mauss and M. D. Robinson, "Measures of emotion: A review," *Cognition and Emotion,* vol. 23, no. 2, pp. 209-237, 2009/02/01 2009, doi: 10.1080/02699930802204677.

[2] L. ZIRUI, "EEG-based emotion recognition using machine learning techniques," *SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING doctor Thesis,* 2018.

[3] F. Noroozi, M. Marjanovic, A. Njegus, S. Escalera, and G. Anbarjafari, "Audio-Visual Emotion Recognition in Video Clips," *IEEE Transactions on Affective Computing,* vol. 10, no. 1, pp. 60-75, 2019, doi: 10.1109/TAFFC.2017.2713783.

[4] S. Jerritta, M. Murugappan, R. Nagarajan, and K. Wan, "Physiological signals based human emotion Recognition: a review," in *2011 IEEE 7th International Colloquium on Signal Processing and its Applications*, 4-6 March 2011 2011, pp. 410-415, doi: 10.1109/CSPA.2011.5759912.

[5] T. Evgeniou and M. Pontil, *Support Vector Machines: Theory and Applications*. 2001, pp. 249-257.

[6] L. E. Peterson, "K-nearest neighbor," *Scholarpedia,* vol. 4, p. 1883, 2009. [Online]. Available: http://www.scholarpedia.org/article/K-nearest_neighbor.

[7] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*, 21-23 Aug. 2017 2017, pp. 1-6, doi: 10.1109/ICEngTechnol.2017.8308186.

[8] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.,* vol. 9, no. 8, pp. 1735–1780, 1997, doi: 10.1162/neco.1997.9.8.1735.

[9] S. Dodge and L. Karam, "Understanding how image quality affects deep neural networks," in *2016 Eighth International Conference on Quality of Multimedia Experience (QoMEX)*, 6-8 June 2016 2016, pp. 1-6, doi: 10.1109/QoMEX.2016.7498955.

[10] X. Xing, Z. Li, T. Xu, L. Shu, B. Hu, and X. Xu, "SAE+LSTM: A New Framework for Emotion Recognition From Multi-Channel EEG," (in English), *Frontiers in Neurorobotics,* Original Research vol. 13, no. 37, 2019-June-12 2019, doi: 10.3389/fnbot.2019.00037.

[11] M. Developers. "MNE." https://mne.tools/stable/index.html (accessed.

[12] S. Koelstra *et al.*, "DEAP: A Database for Emotion Analysis ;Using Physiological Signals," *IEEE Transactions on Affective Computing,* vol. 3, no. 1, pp. 18-31, 2012, doi: 10.1109/T-AFFC.2011.15.

[13] B. JW, F. LC, and H. J. e. al., "Electroencephalography (EEG): An Introductory Text and Atlas of Normal and Abnormal Findings in Adults, Children, and Infants

[Internet]." *Chicago: American Epilepsy Society; 2016.,* vol. Appendix 6. A Brief History of EEG., p. https://www.ncbi.nlm.nih.gov/books/NBK390348/.

[14] K. Böcker, J. Avermaete, and M. Berg-Lenssen, "The international 10–20 system revisited: Cartesian and spherical co-ordinates," *Brain topography,* vol. 6, pp. 231-5, 02/01 1994, doi: 10.1007/BF01187714.

[15] S. Suurmets. "Neural Oscillations – Interpreting EEG Frequency Bands." https://imotions.com/blog/neural-oscillations/ (accessed.

[16] M. A. Lexa, M. E. Davies, J. S. Thompson, and J. Nikolic, "Compressive power spectral density estimation," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 22-27 May 2011 2011, pp. 3884-3887, doi: 10.1109/ICASSP.2011.5947200.

[17] P. Welch, "The use of fast Fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms," *IEEE Transactions on Audio and Electroacoustics,* vol. 15, no. 2, pp. 70-73, 1967, doi: 10.1109/TAU.1967.1161901.

[18] A. Mehrabian, "Pleasure-arousal-dominance: A general framework for describing and measuring individual differences in Temperament," *Current Psychology,* vol. 14, no. 4, pp. 261-292, 1996/12/01 1996, doi: 10.1007/BF02686918.

[19] R. A. Stevenson and T. W. James, "Affective auditory stimuli: Characterization of the International Affective Digitized Sounds (IADS) by discrete emotional categories," *Behavior Research Methods,* vol. 40, no. 1, pp. 315-321, 2008/02/01 2008, doi: 10.3758/BRM.40.1.315.

[20] W. Zheng and B. Lu, "Investigating Critical Frequency Bands and Channels for EEG-Based Emotion Recognition with Deep Neural Networks," *IEEE Transactions on Autonomous Mental Development,* vol. 7, no. 3, pp. 162-175, 2015, doi: 10.1109/TAMD.2015.2431497.

[21] B. Richhariya and M. Tanveer, "EEG signal classification using universum support vector machine," *Expert Systems with Applications,* vol. 106, pp. 169-182, 2018/09/15/ 2018, doi: https://doi.org/10.1016/j.eswa.2018.03.053.

[22] Z. Wei, C. Wu, X. Wang, A. Supratak, P. Wang, and Y. Guo, "Using Support Vector Machine on EEG for Advertisement Impact Assessment," (in English), *Frontiers in Neuroscience,* Original Research vol. 12, no. 76, 2018-March-12 2018, doi: 10.3389/fnins.2018.00076.

[23] H. Candra *et al.*, "Investigation of Window Size in Classification of EEG-Emotion Signal with Wavelet Entropy and Support Vector Machine," *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC),* vol. pp. 7250-7253, 2015.

[24] X. Lun, Z. Yu, T. Chen, F. Wang, and Y. Hou, "A Simplified CNN Classification Method for MI-EEG via the Electrode Pairs Signals," (in English), *Frontiers in Human Neuroscience,* Methods vol. 14, no. 338, 2020-September-15 2020, doi: 10.3389/fnhum.2020.00338.

[25] Y. Gao, B. Gao, Q. Chen, J. Liu, and Y. Zhang, "Deep Convolutional Neural Network-Based Epileptic Electroencephalogram (EEG) Signal Classification," (in English), *Frontiers in Neurology,* Methods vol. 11, no. 375, 2020-May-22 2020, doi: 10.3389/fneur.2020.00375.

[26] J. X. Chen, P. W. Zhang, Z. J. Mao, Y. F. Huang, D. M. Jiang, and Y. N. Zhang, "Accurate EEG-Based Emotion Recognition on Combined Features Using Deep Convolutional Neural Networks," *IEEE Access,* vol. 7, pp. 44317-44328, 2019, doi: 10.1109/ACCESS.2019.2908285.

[27] S. Liu, X. Wang, L. Zhao, J. Zhao, Q. Xin, and S. Wang, "Subject-independent Emotion Recognition of EEG Signals Based on Dynamic Empirical Convolutional Neural Network," *IEEE/ACM Transactions on Computational Biology and Bioinformatics,* pp. 1-1, 2020, doi: 10.1109/TCBB.2020.3018137.

[28] R. Pascanu, T. Mikolov, and Y. Bengio, "Understanding the exploding gradient problem," *ArXiv,* vol. abs/1211.5063, 2012.

[29] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," *INTERSPEECH-2014,* vol. 338-342, 2014.

[30] X. Hu, S. Yuan, F. Xu, Y. Leng, K. Yuan, and Q. Yuan, "Scalp EEG classification using deep Bi-LSTM network for seizure detection," *Computers in Biology and Medicine,* vol. 124, p. 103919, 2020/09/01/ 2020, doi: https://doi.org/10.1016/j.compbiomed.2020.103919.

[31] S. A. a. A. A. F. a. R. A. El-Khoribi, "Emotion Recognition based on EEG using LSTM Recurrent Neural Network," *International Journal of Advanced Computer Science and Applications,* vol. 8, doi: 10.14569/IJACSA.2017.081046.

[32] G. E. Hinton and R. R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," *Science,* vol. 313, no. 5786, p. 504, 2006, doi: 10.1126/science.1127647.

[33] D. Charte, F. Charte, M. J. del Jesus, and F. Herrera, "An analysis on the use of autoencoders for representation learning: Fundamentals, learning task case studies, explainability and challenges," *Neurocomputing,* vol. 404, pp. 93-107, 2020/09/03/ 2020, doi: https://doi.org/10.1016/j.neucom.2020.04.057.

[34] B. Yang, X. Han, and J. Tang, "Three class emotions recognition based on deep learning using staked autoencoder," in *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, 14-16 Oct. 2017 2017, pp. 1-5, doi: 10.1109/CISP-BMEI.2017.8302098.

[35] T. Song, W. Zheng, P. Song, and Z. Cui, "EEG Emotion Recognition Using Dynamical Graph Convolutional Neural Networks," *IEEE Transactions on Affective Computing,* vol. 11, no. 3, pp. 532-541, 2020, doi: 10.1109/TAFFC.2018.2817622.

[36] X. Wang, T. Zhang, X. Xu, L. Chen, X. Xing, and C. L. P. Chen, "EEG Emotion Recognition Using Dynamical Graph Convolutional Neural Networks and Broad

Learning System," in *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 3-6 Dec. 2018 2018, pp. 1240-1244, doi: 10.1109/BIBM.2018.8621147.

[37] X. Chai, Q. Wang, Y. Zhao, X. Liu, O. Bai, and Y. Li, "Unsupervised domain adaptation techniques based on auto-encoder for non-stationary EEG-based emotion recognition," *Computers in Biology and Medicine,* vol. 79, pp. 205-214, 2016/12/01/ 2016, doi: https://doi.org/10.1016/j.compbiomed.2016.10.019.

[38] X. Chai *et al.*, "A Fast, Efficient Domain Adaptation Technique for Cross-Domain Electroencephalography(EEG)-Based Emotion Recognition," (in eng), *Sensors (Basel),* vol. 17, no. 5, p. 1014, 2017, doi: 10.3390/s17051014.

[39] Y. Li, J. Huang, H. Zhou, and N. Zhong, "Human Emotion Recognition with Electroencephalographic Multidimensional Features by Hybrid Deep Neural Networks," *Applied Sciences,* vol. 7, no. 10, 2017, doi: 10.3390/app7101060.

[40] P. Zhong, D. Wang, and C. Miao, "EEG-Based Emotion Recognition Using Regularized Graph Neural Networks," *IEEE Transactions on Affective Computing,* 2020, doi: DOI 10.1109/TAFFC.2020.2994159.

Appendix

GitHub URL:

https://github.com/Jacob12138xieyuan/EEG-Based-Emotion-Recognition-on-DEAP

```
from IPython.utils import io
import numpy as np
import collections

from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.utils import shuffle

import scipy.io
from scipy import signal, integrate
import matplotlib.pyplot as plt

import keras
from keras.models import Model
from keras.layers import Input, Dense, LSTM, Dropout

import mne
import eeg_entropy
import math

n_second = 60
n_segment = 2*n_second-1
n_points = n_second*128
bottleneck = 12

# build in
a = np.array([[1,2,3],[4,5,6],[7,8,9]])
scaler = MinMaxScaler().fit(a)
a = scaler.transform(a)
print(a)

#custom
a = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

```
max_, min_ = np.amax(a), np.amin(a)
a = (a - min_) / (max_ - min_)
print(a)

def normalise_2D(a, multiple):
	max_, min_ = np.amax(a), np.amin(a)
	a = (a - min_) / (max_ - min_)
	return multiple*a

a = np.array([[1,2,3],[4,5,6],[7,8,9]])
normalise_2D(a, 1) # range -10 to 10

# build in
a = np.array([[1,2,3],[4,5,6],[7,8,9]])
scaler = StandardScaler().fit(a)
a = scaler.transform(a)
print(a)

a = np.array([[1,2,3],[4,5,6],[7,8,9]])
a = np.transpose(a)
scaler = StandardScaler().fit(a)
a = scaler.transform(a)
a = np.transpose(a)
print(a)

# custom
a = np.array([[1,2,3],[4,5,6],[7,8,9]])
std = np.std(a)
mean = np.mean(a)
a = (a-mean)/std
print(a)

def standardise_2D(a, multiple):
	std = np.std(a)
	mean = np.mean(a)
	a = (a-mean)/std
	return multiple*a

a = np.array([[1,2,3],[4,5,6],[7,8,9]])
standardise_2D(a, 1)
```

```
def convertOneData(file_name):
    mat = scipy.io.loadmat(file_name)
    labels = mat['labels'][:, 0:2] # only valence, arousal, no dominance, liking
    data = mat['data'][:, 0:32, 3*128:] # only first 32 channels ['Fp1','AF3','F3','F7','FC5',
    #'FC1','C3','T7','CP5','CP1','P3','P7','PO3','O1','Oz','Pz','Fp2','AF4','Fz','F4',
    #'F8','FC6','FC2','Cz','C4','T8','CP6','CP2','P4','P8','PO4','O2']
    #and skip first 3 seconds
    #print(labels.shape, data.shape) # (40, 2) (40, 32, 8064)
    valence_labels, valence_data = [], []
    arousal_labels, arousal_data = [], []
    for i, label in enumerate(labels):
        valence, arousal = label[0], label[1]
        if valence > 5.5: # value >5.5 is high
            valence_labels.append(1)
            valence_data.append(data[i])
        if valence < 4.5: # value <4.5 is high
            valence_labels.append(0)
            valence_data.append(data[i])
        if arousal > 5.5:
            arousal_labels.append(1)
            arousal_data.append(data[i])
        if arousal < 4.5:
            arousal_labels.append(0)
            arousal_data.append(data[i])

    print("valence: ", len(valence_labels), "arousal: ", len(arousal_labels))
    return valence_labels, valence_data, arousal_labels, arousal_data

def convertAllData():
    all_valence_labels, all_valence_data = [], []
    all_arousal_labels, all_arousal_data = [], []
    for i in range(32):
        if i < 10: # subject 01-09
            name = '%0*d' % (2,i+1)
        else: # subject 10-32
            name = i+1
        file_name = "../Data/DEAP/s"+str(name)+".mat"
        print(file_name)
        valence_labels, valence_data, arousal_labels, arousal_data =
convertOneData(file_name) # convert one subject data
```

```
            all_valence_labels += valence_labels
            for valence_d in valence_data: # each trial
                valence_d = standardise_2D(valence_d, 1)
                all_valence_data.append(valence_d)
            all_arousal_labels += arousal_labels
            for arousal_d in arousal_data:
                arousal_d = standardise_2D(arousal_d, 1)
                all_arousal_data.append(arousal_d)

    all_valence_labels = np.array(all_valence_labels)
    all_valence_data = np.array(all_valence_data)
    all_arousal_labels = np.array(all_arousal_labels)
    all_arousal_data = np.array(all_arousal_data)
    print("Valence trial data for all subject: ",
all_valence_labels.shape,all_valence_data.shape)
    print("Arousal trial data for all subject: ",
all_arousal_labels.shape,all_arousal_data.shape)
    # save numpy array of total data to files
    np.save('../Data/processed_DEAP/valence/' + 'all_valence_labels.npy',
all_valence_labels)
    np.save('../Data/processed_DEAP/valence/' + 'all_valence_data.npy',
all_valence_data)
    np.save('../Data/processed_DEAP/arousal/' + 'all_arousal_labels.npy',
all_arousal_labels)
    np.save('../Data/processed_DEAP/arousal/' + 'all_arousal_data.npy',
all_arousal_data)

convertAllData()

def load_np_data(dimension):
    if dimension == 'valence':
        all_labels, all_data = np.load('../Data/processed_DEAP/valence/' +
'all_valence_labels.npy', allow_pickle=True), np.load('../Data/processed_DEAP/valence/'
+ 'all_valence_data.npy', allow_pickle=True)
        print("Total valence: ", all_labels.shape, all_data.shape)
        #print("High and low valence: ", collections.Counter(all_labels))# 587 high
valence, 472 low valence
    elif dimension == 'arousal':
        all_labels, all_data = np.load('../Data/processed_DEAP/arousal/' +
'all_arousal_labels.npy', allow_pickle=True), np.load('../Data/processed_DEAP/arousal/'
+ 'all_arousal_data.npy', allow_pickle=True)
```

```
            print("Total arousal: ", all_labels.shape, all_data.shape)
            #print("High and low arousal: ", collections.Counter(all_labels))# 620 high
arousal, 462 low arousal
    return all_labels, all_data

#all_labels, all_data = load_np_data(dimension="valence")
all_labels, all_data = load_np_data(dimension="arousal")

def trial_psd_extraction_integration(data): # data shape (12, 8064)
    info = mne.create_info(ch_names= ['1','2','3','4','5','6','7','8','9','10','11','12'],
sfreq=128);
    raw = mne.io.RawArray(data, info, first_samp=0, copy='auto', verbose=None);
    psd_origin, f = mne.time_frequency.psd_welch(raw, fmin=0, fmax=60, n_fft=128,
n_overlap=64, n_per_seg=128, picks='all', window='hann', average=None,
verbose=None)# average='mean' or None
    # print(psd_origin.shape, f.shape) # (12, 61, 119) (61,) 61 frequency
    psd = np.moveaxis(psd_origin, -1, 0) # (119, 12, 61)
    # calculate frequency band power using integration
    band_power = [] # band power for all segments
    for segment in psd:
        segment_band_power = [] # band power for all channels in one segment
        for psd_channel in segment:
            y_int = integrate.cumtrapz(psd_channel, f, initial=0) # integrate to
calculate band power
            one_band_power = np.array([y_int[7]-y_int[4],y_int[13]-
y_int[8],y_int[30]-y_int[14],y_int[51]-y_int[31]])
            segment_band_power.append(one_band_power)
        band_power.append(segment_band_power)
    band_power = np.array(band_power) # (119, 12, 4)
    band_power = np.moveaxis(band_power, -1, 1) # (119, 4, 12)
    band_power = band_power.reshape((n_segment, bottleneck*4)) # flatten feature
(119, 48)
    band_power = 10*band_power
    return band_power

# change dimension from (849, 32, 8064)to (849, 8064, 32) then to (6846336, 32) for
input 32-dimension vector to autoencoder
def vector_transform(data):
    vectors = np.moveaxis(data, 1, -1)
    vectors = vectors.reshape((vectors.shape[0]*vectors.shape[1], vectors.shape[2]))
    return vectors
```

```
# change output of autoencoder dimension from (6846336, 12) to (849, 8064, 12) then to
(849, 12, 8064)
def inverse_vector_transform(vectors):
	data = vectors.reshape((int(vectors.shape[0]/n_points), n_points, vectors.shape[1]))
	data = np.moveaxis(data, -1, 1)
	return data

all_data, all_labels = shuffle(all_data, all_labels, random_state=0)
n = len(all_labels) # 1059
print(n)
fold_n = math.floor(n/10) # 105
print(fold_n)
all_data, all_labels = all_data[:10*fold_n], all_labels[:10*fold_n] # (1050, 32, 8064)
print(all_data.shape)

def process(test_fold_number):
	# train has 9 folds, test has 1 fold
	train_data = np.concatenate((all_data[:test_fold_number*fold_n],
all_data[fold_n+test_fold_number*fold_n:]), axis=0)
	train_labels = np.concatenate((all_labels[:test_fold_number*fold_n],
all_labels[fold_n+test_fold_number*fold_n:]), axis=0)
	test_data = all_data[test_fold_number*fold_n : fold_n+test_fold_number*fold_n]
	test_labels = all_labels[test_fold_number*fold_n :
fold_n+test_fold_number*fold_n]
	print(train_data.shape,test_data.shape) # (945, 32, 8064) (105, 32, 8064)

	# change dimension to 32-dimension vector for input to autoencoder
	train_vectors = vector_transform(train_data)
	test_vectors = vector_transform(test_data)
	print(train_vectors.shape, test_vectors.shape)# (7620480, 32) (846720, 32)

	# -------- Create new autoencoder --------
	input_layer = Input(shape=(32,))
	encoded = Dense(64, activation=None)(input_layer)
	bottleneck_layer = Dense(bottleneck, activation=None)(encoded)
	decoded = Dense(64, activation=None)(bottleneck_layer)
	decoded = Dense(32, activation=None)(decoded)
	autoencoder = Model(input_layer, decoded)
	#autoencoder.summary()
```

```
    encoder = Model(input_layer, bottleneck_layer)
    #encoder.summary()

    decoder_input_layer = Input(shape=(bottleneck,))
    decoder_layer = autoencoder.layers[-2](decoder_input_layer)
    decoder_layer = autoencoder.layers[-1](decoder_layer)
    decoder = Model(decoder_input_layer, decoder_layer)
    #decoder.summary()

    # -------- Compile and train autoencoder --------
    autoencoder.compile(optimizer='SGD', loss='mse', metrics=['accuracy'])
    autoencoder.fit(train_vectors, train_vectors, epochs=1, batch_size=64, shuffle=True,
validation_data=(test_vectors, test_vectors))
    autoencoder.save("../Results/autoencoder_model/autoencoder_model_test_fold_" +
str(test_fold_number))

    # -------- Encode train and test data by pass through encoder --------
    train_data_encoded = encoder.predict(train_vectors)
    train_data_encoded = inverse_vector_transform(train_data_encoded)
    test_data_encoded = encoder.predict(test_vectors)
    test_data_encoded = inverse_vector_transform(test_data_encoded)
    print("Encoded training data shape: ", train_data_encoded.shape)
    print("Encoded test data shape: ", test_data_encoded.shape)

    # -------- Feature extraction from 12 source signal --------
    train_band_power = [] # band power feature sequence for train trials
    for data in train_data_encoded: # for every train trial
        with io.capture_output() as captured:
            trial_band_power = trial_psd_extraction_integration(data) # data shape
(12, 8064)
        train_band_power.append(trial_band_power)
    train_band_power = np.array(train_band_power)

    test_band_power = [] # band power feature sequence for test trials
    for data in test_data_encoded: # for every test trial
        with io.capture_output() as captured:
            trial_band_power = trial_psd_extraction_integration(data) # data shape
(12, 8064)
        test_band_power.append(trial_band_power)
    test_band_power = np.array(test_band_power)
```

```
    print("All features of training data shape: ", train_band_power.shape) # shape (849,
119, 48)
    print("All features of test data shape: ", test_band_power.shape) # shape (95, 119,
48)

    # -------- Create new LSTM model --------
    x=Input(shape=(n_segment,bottleneck*4)) # flatten (12,4) to 48
    x1=LSTM(n_segment)(x)
    x2=Dense(n_segment)(x1)
    x3=Dense(12)(x2)
    output=Dense(1, activation="sigmoid")(x2)
    model=Model(x, output)

    # -------- Compile and train LSTM --------
    model.compile(optimizer='SGD', loss='mse', metrics=['accuracy'])
    history = model.fit(train_band_power, train_labels, epochs=30, batch_size=8,
validation_data=(test_band_power, test_labels))
    print("Hightest accuracy: " + str(max(history.history['val_accuracy'])))
    model.save("../Results/LSTM_model/LSTM_model_test_fold_" +
str(test_fold_number))

for i in range(10):
    print("********** Test Fold " + str(i) + " **********")
    process(i)

# extract feature from 12-channel source signal
data = train_data_encoded[0]
info = mne.create_info(ch_names= ['1','2','3','4','5','6','7','8','9','10','11','12',], sfreq=128)
raw = mne.io.RawArray(data, info, first_samp=0, copy='auto', verbose=None)
raw.plot(duration=3, n_channels=12);

# calculate PSD of all channels using welch method
psd_origin, f = mne.time_frequency.psd_welch(raw, fmin=0, fmax=60, n_fft=128,
n_overlap=64, n_per_seg=128, picks='all', window='hann', average=None,
verbose=None)# average='mean' or None
print(psd_origin.shape, f.shape) # (12, 61, 119) (61,)
plt.plot(f, psd_origin[0]) # first channel psd of 119 segments
plt.title("PSD feature for 119 segments of first channel")
plt.xlabel('frequency [Hz]')
plt.ylabel('PSD')
plt.show()
```

```
mat = scipy.io.loadmat('../Data/DEAP/s01.mat')
data = mat['data'][:, 0:32, 3*128:] # only first 32 channels
['Fp1','AF3','F3','F7','FC5','FC1','C3','T7','CP5','CP1','P3','P7','PO3','O1','Oz','Pz','Fp2','AF
4','Fz','F4','F8','FC6','FC2','Cz','C4','T8','CP6','CP2','P4','P8','PO4','O2']

one_data = data[0]

# change to mne format # system used in DEAP 'biosemi32'
biosemi32 = mne.channels.make_standard_montage('biosemi32')
info = mne.create_info(ch_names=biosemi32.ch_names, ch_types='eeg', sfreq=128)
raw = mne.EvokedArray(one_data, info) # first trial evoked data

print(data.shape)
print(np.amax(data)) # max value
print(np.amin(data)) # min value

info_ = mne.create_info(ch_names=
['Fp1','AF3','F3','F7','FC5','FC1','C3','T7','CP5','CP1','P3','P7','PO3','O1','Oz','Pz','Fp2','AF
4','Fz','F4','F8','FC6','FC2','Cz','C4','T8','CP6','CP2','P4','P8','PO4','O2'], sfreq=128)
raw_ = mne.io.RawArray(one_data, info_, first_samp=0, copy='auto', verbose=None);
raw_.plot(duration=3, n_channels=20);

biosemi32.plot(kind='topomap', show_names=True);

raw = raw.set_montage(biosemi32)
mne.viz.plot_topomap(raw.data[:, 0], raw.info, show=False) # time step 0

times = np.arange(30.01, 30.10, 0.01) # from 30s to 30.2s
raw.plot_topomap(times, ch_type='eeg', time_unit='s', ncols=5, nrows='auto');

standardised_data = standardise_2D(one_data, 1)

print(np.amax(one_data)) # max value
print(np.amin(one_data)) # min value
print(np.amax(standardised_data)) # max value
print(np.amin(standardised_data)) # min value

nth = 10
plt.figure(figsize=(15, 3))
plt.plot(one_data[0][nth*128:nth*128+128]) # first trial first channel
```

```
plt.title("Original signal")
plt.xlabel('Points')
plt.ylabel('value')
plt.show()

f, Pxx_den = signal.welch(one_data[0][nth*128:nth*128+128], fs=128, window='hann',
nperseg = 128, noverlap=64)
plt.plot(f, Pxx_den)
plt.ylim()
plt.xlabel('frequency [Hz]')
plt.ylabel('PSD')
plt.show()

plt.figure(figsize=(15, 3))
plt.plot(standardised_data[0][nth*128:nth*128+128])
plt.title("Standardised signal")
plt.xlabel('Points')
plt.ylabel('value')
plt.show()

f, Pxx_den = signal.welch(standardised_data[0][nth*128:nth*128+128], fs=128,
window='hann', nperseg = 128, noverlap=64)
plt.plot(f, Pxx_den)
plt.ylim()
plt.xlabel('frequency [Hz]')
plt.ylabel('PSD')
plt.show()

channel_psd, channel_psd2 =
eeg_entropy.bin_power(standardised_data[0][nth*128:nth*128+128],
[4,7.5,13.5,30.5,50], Fs=128)
print(channel_psd)

info2 = mne.create_info(ch_names=
['Fp1','AF3','F3','F7','FC5','FC1','C3','T7','CP5','CP1','P3','P7','PO3','O1','Oz','Pz','Fp2','AF
4','Fz','F4','F8','FC6','FC2','Cz','C4','T8','CP6','CP2','P4','P8','PO4','O2'], sfreq=128)
raw2 = mne.io.RawArray(one_data, info2, first_samp=0, copy='auto', verbose=None);
raw2.plot(duration=3, n_channels=20);

stanardised_raw2 = mne.io.RawArray(standardised_data, info2, first_samp=0,
copy='auto', verbose=None);
```

```
stanardised_raw2.plot(duration=3, n_channels=20);

def normalise_2D(a, multiple):
    max_, min_ = np.amax(a), np.amin(a)
    a = (a - min_) / (max_ - min_)
    return multiple*a

normalised_data = normalise_2D(one_data, 1)

nth = 10
plt.figure(figsize=(15, 3))
plt.plot(one_data[0][nth*128:nth*128+128]) # first trial first channel
plt.title("Original signal")
plt.xlabel('Points')
plt.ylabel('value')
plt.show()

f, Pxx_den = signal.welch(one_data[0][nth*128:nth*128+128], fs=128, window='hann',
nperseg = 128, noverlap=64)
plt.plot(f, Pxx_den)
plt.ylim()
plt.xlabel('frequency [Hz]')
plt.ylabel('PSD')
plt.show()

plt.figure(figsize=(15, 3))
plt.plot(normalised_data[0][nth*128:nth*128+128])
plt.title("Normalised signal")
plt.xlabel('Points')
plt.ylabel('value')
plt.show()

f, Pxx_den = signal.welch(normalised_data[0][nth*128:nth*128+128], fs=128,
window='hann', nperseg = 128, noverlap=64)
plt.plot(f, Pxx_den)
plt.ylim()
plt.xlabel('frequency [Hz]')
plt.ylabel('PSD')
plt.show()
raw3 = mne.io.RawArray(one_data, info2, first_samp=0, copy='auto', verbose=None);
raw3.plot(duration=3, n_channels=20);
```

```
normalised_raw3 = mne.io.RawArray(normalised_data, info2, first_samp=0, copy='auto',
verbose=None);
normalised_raw3.plot(duration=3, n_channels=20);
```