B.Tech. BCSE497J - Project-I

Advanced Emotion Recognition from EEG: A Hybrid Approach Using Autoencoders and LSTM

21BCI0365	**ADHIRAJ SINGH CHEEMA**
21BCT0066	**AARON MANO CHERIAN**
21BCE3904	**NIDHI VASANT BHAT**

Under the Supervision of

Natarajan P

Professor

School of Computer Science and Engineering (SCOPE)

B.Tech.

in

Computer Science and Engineering

(with specialization in Information Security)

School of Computer Science and Engineering

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

November 2024

DECLARATION

I hereby declare that the project entitled "**Advanced Emotion Recognition from EEG: A Hybrid Approach Using Autoencoders and LSTM"** submitted by me, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering* to VIT is a record of bonafide work carried out by me under the supervision of Prof. Dr. Natarajan P.

I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place : Vellore

Date : 19/11/2024

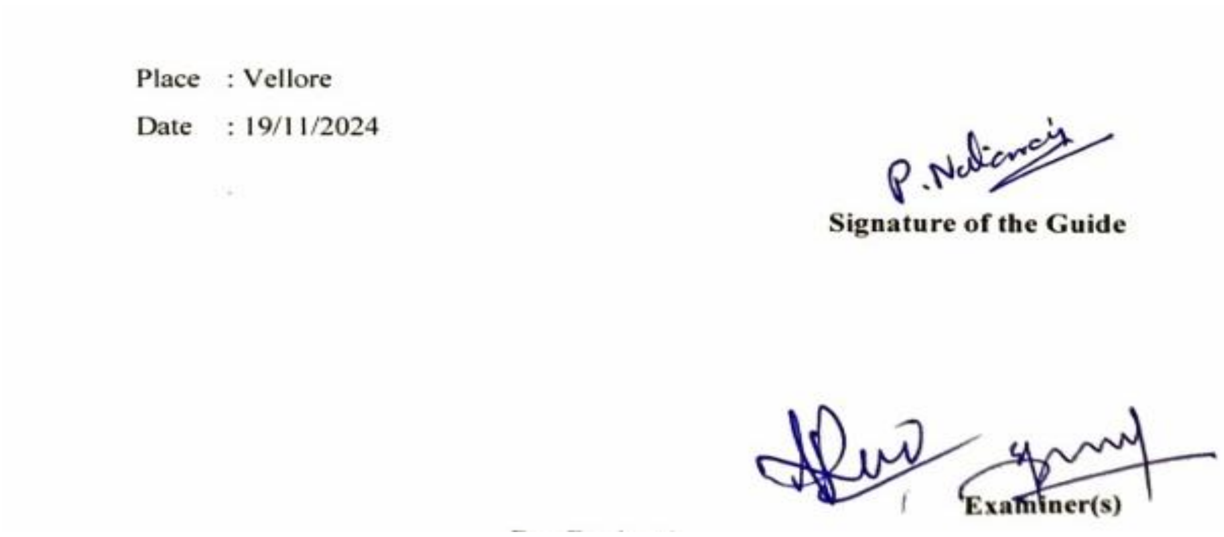
[illegible]

Signature of the Candidate

<u>CERTIFICATE</u>

This is to certify that the project entitled "**Advanced Emotion Recognition from EEG: A Hybrid Approach Using Autoencoders and LSTM"** submitted by **ADHIRAJ SINGH CHEEMA (21BCI0365), NIDHI VASANT BHAT (21BCE3904) & AARON MANO CHERIAN (21BCT006), School of Computer Science and Engineering**, VIT, for the award of the degree of *Bachelor of Technology in Computer Science and Engineering,* is a record of bonafide work carried out by him / her under my supervision during Fall Semester 2024-2025, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The project fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.



Place : Vellore

Date : 19/11/2024

[illegible]

Signature of the Guide

[illegible]

Examiner(s)

Dr. Sharmila Banu

Bachelors of Technology in Computer Science Engineering

ACKNOWLEDGEMENTS

I am deeply grateful to the management of Vellore Institute of Technology (VIT) for providing me with the opportunity and resources to undertake this project. Their commitment to fostering a conducive learning environment has been instrumental in my academic journey. The support and infrastructure provided by VIT have enabled me to explore and develop my ideas to their fullest potential.

My sincere thanks to Dr. Ramesh Babu K, the Dean of the School of Computer Science and Engineering (SCOPE), for his unwavering support and encouragement. His leadership and vision have greatly inspired me to strive for excellence. The Dean's dedication to academic excellence and innovation has been a constant source of motivation for me. I appreciate his efforts in creating an environment that nurtures creativity and critical thinking.

I express my profound appreciation to Dr. Sharmila Banu, the Head of the Internet of Things Department, for his/her insightful guidance and continuous support. His/her expertise and advice have been crucial in shaping the direction of my project. The Head of Department's commitment to fostering a collaborative and supportive atmosphere has greatly enhanced my learning experience. His/her constructive feedback and encouragement have been invaluable in overcoming challenges and achieving my project goals.

I am immensely thankful to my project supervisor, Dr. Natarajan P, for his/her dedicated mentorship and invaluable feedback. His/her patience, knowledge, and encouragement have been pivotal in the successful completion of this project. My supervisor's willingness to share his/her expertise and provide thoughtful guidance has been instrumental in refining my ideas and methodologies. His/her support has not only contributed to the success of this project but has also enriched my overall academic experience.

Thank you all for your contributions and support.

Aaron Mano Cherian

TABLE OF CONTENTS

ABSTRACT

Emotion recognition based on electroencephalogram (EEG) signals has emerged as a crucial technology for enhancing human-computer interaction, with wide-ranging applications in healthcare, entertainment, and psychology. EEG-based methods are increasingly preferred over traditional facial or auditory recognition techniques due to their superior accuracy and reliability. While subject-dependent emotion recognition has been extensively studied using conventional machine learning models, recent findings indicate that deep learning approaches excel in subject-independent emotion classification.

This report presents an advanced hybrid approach to EEG-based emotion recognition, combining autoencoders and Long Short-Term Memory (LSTM) networks. Our novel architecture leverages autoencoders to decompose raw EEG data into key signal components, from which we extract power spectral density (PSD) features. The temporal dynamics of these PSD feature sequences are then captured using an LSTM recurrent neural network, enabling robust emotion classification.

Extensive experiments were conducted to optimize the model structure and hyperparameters, ensuring peak performance of our hybrid approach. Additionally, we employ the open-source Python packages MNE and SciPy for comprehensive EEG data analysis, visualization, and interpretation.

This research contributes to the field of affective computing by introducing a powerful hybrid model that combines the strengths of autoencoders for feature extraction and LSTM for temporal sequence learning. Our work paves the way for more advanced, reliable EEG-based emotion recognition systems, with potential implications for personalized healthcare, adaptive user interfaces, and a deeper understanding of human emotional processes.

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. 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. 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. 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 . 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) , k-nearest neighbor (KNN) 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) , Long Short-Term model (LSTM) , Dense Neural Network (DNN) , and so on. In this report, 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) 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 . In the report, the DEAP 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 Motivation

The rapid advancement of modern science and technology has brought increased attention to the study of human emotions, which reflect one's mental state and affective reactions based on subjective experiences. Emotion plays a crucial role not only in human-to-human communication but also in various industries such as healthcare, gaming, and psychology. As such, there is a growing need for developing effective emotion recognition algorithms that enable computers to accurately identify and understand human emotions, thereby enhancing human-computer interaction.

Traditionally, emotion recognition has relied primarily on visual and auditory cues, such as facial expressions and voice analysis. These methods mirror how humans typically perceive emotions in others. Consequently, cameras and microphones have been widely deployed as the "eyes and ears" of computers, collecting image and audio data for emotion classification tasks. However, these surface-level features have significant limitations. For instance, in low-light conditions like movie theaters, cameras may struggle to capture facial expressions accurately. Moreover, research has shown that facial expressions don't always align with true emotional states, as observed in cinema studies where audiences maintained neutral expressions despite experiencing emotional changes induced by the film content.

These limitations have spurred research into alternative methods that can more directly detect human affective states, particularly through physiological signals. Among various physiological signals, Electroencephalogram (EEG) has emerged as one of the most promising for emotion analysis. EEG records electrical signals induced by brain activities, measurable from the scalp surface using dry electrodes. The increasing popularity of portable Brain-Computer Interface (BCI) devices, coupled with a desire for deeper understanding of human brain activities, has made EEG-based emotion recognition a focal point of research interest.

In recent years, researchers have intensively studied various features of EEG signals, including time domain, frequency domain, and time-frequency domain characteristics. Traditional machine learning methods such as Support Vector Machines (SVM) and k-Nearest Neighbors (KNN) have been applied to classify different emotions based on these features. With appropriate EEG data pre-processing and model hyperparameter tuning, satisfactory results have been achieved, particularly on a subject-dependent basis.

However, the subject-dependent nature of these models presents a significant challenge. Due to the inherent differences in EEG patterns among individuals, models trained on one subject often fail to generalize well to others. This limitation severely restricts the practical applicability of EEG-based emotion recognition systems. The holy grail of this field is to achieve subject-independent emotion recognition, where a single model can accurately predict emotions across different individuals.

The advent of deep learning techniques has opened new avenues for addressing this challenge. Novel architectures and creative algorithms leveraging Convolutional Neural Networks (CNN), Long Short-Term Memory networks (LSTM), and Dense Neural Networks (DNN) have shown promise in capturing complex patterns in EEG data. These approaches have the potential to uncover deeper relationships between EEG signals and emotional states across diverse subjects.

Particularly intriguing are hybrid approaches that combine multiple deep learning techniques. For instance, the SAE+LSTM architecture, which integrates Stacked Autoencoders with LSTM networks, has demonstrated encouraging results in recent studies. Such hybrid models can potentially leverage the strengths of different neural network architectures to better capture both spatial and temporal aspects of EEG signals.

Despite these advancements, a fully subject-independent emotion recognition algorithm using EEG data remains elusive. This gap in the current state of the art serves as a primary motivation for our research. We aim to explore the in-depth relationship between EEG data and emotions across multiple subjects using various deep learning techniques. By leveraging tools like the Python package MNE for EEG signal visualization and analysis, we hope to gain deeper insights into the nature of these signals and their correlation with emotional states.

Our study will focus on the DEAP benchmark dataset, aiming to perform subject-independent binary emotion classification on both valence and arousal dimensions. By adhering to the 10-fold cross-validation standard used in previous research, we ensure comparability with existing studies while pushing the boundaries of what's possible in EEG-based emotion recognition.

The potential impact of this research extends far beyond academic interest. Successful development of subject-independent EEG-based emotion recognition could revolutionize human-computer interaction across various domains. In healthcare, it could lead to more accurate diagnosis and monitoring of mental health conditions. In gaming and entertainment, it could enable more immersive and responsive experiences. In psychology and neuroscience, it could provide new tools for understanding the complexities of human emotions and cognition.

In essence, our research is driven by the need to overcome the limitations of current emotion recognition techniques, harness the full potential of EEG signals, and address the fundamental challenge of subject-independent emotion recognition. Through the application of advanced deep learning approaches and rigorous analysis, we aim to contribute to the development of more accurate, robust, and generalizable emotion recognition systems, potentially opening new frontiers in human-computer interaction and emotional intelligence.

1.3 Scope

This research project aims to advance the field of EEG-based emotion recognition by leveraging state-of-the-art deep learning techniques, with a particular focus on achieving subject-independent classification. The cornerstone of our study will be the DEAP (Database for Emotion Analysis using Physiological Signals) dataset, which provides a rich collection of EEG recordings from 32 subjects, each undergoing 40 trials. Our analysis will center on the valence and arousal dimensions of emotion, seeking to develop robust methods for binary classification in these emotional spaces.

The project will begin with a comprehensive examination of the DEAP dataset, including thorough preprocessing to clean and normalize the EEG signals. We will explore a wide array of feature extraction methods, spanning time-domain, frequency-domain, and time-frequency domain approaches. Special attention will be given to Power Spectral Density (PSD) analysis for deriving Frequency Band Power (FBP) features across the standard EEG frequency bands: Delta, Theta, Alpha, Beta, and Gamma. These features will serve as the foundation for our deep learning models.

At the heart of our research will be the development and comparison of several deep learning architectures. We will implement Convolutional Neural Networks (CNNs) to capture spatial features within the EEG signals, Long Short-Term Memory (LSTM) networks to model temporal dependencies, and Dense Neural Networks (DNNs) for high-level feature learning. A key focus will be on hybrid models, particularly the SAE+LSTM (Stacked Autoencoder + LSTM) architecture, which has shown promise in recent literature. Additionally, we will explore the potential of Graph Neural Networks (GNNs) to model the complex inter-channel relationships present in EEG data.

The primary challenge and objective of this project is to achieve subject-independent classification. To this end, we will implement and evaluate various domain adaptation techniques and transfer learning approaches. These methods will be crucial in addressing the significant inter-subject variability inherent in EEG data and moving towards a more generalizable emotion recognition system.

Our evaluation methodology will employ 10-fold cross-validation to ensure robust performance assessment. We will focus on binary classification for both valence and arousal dimensions separately, utilizing standard performance metrics such as accuracy, precision, recall, and F1-score. A comprehensive comparison with state-of-the-art methods reported in the literature will be conducted to contextualize our results within the field.

Visualization and interpretation of our models will be a key component of the project. We will leverage the MNE Python package for EEG signal visualization and develop techniques to interpret the learned features and decision-making processes of our deep learning models. This will not only aid in understanding the relationship between EEG patterns and emotional states but also provide insights that could guide future research in the field.

The project will also involve systematic hyperparameter optimization for each deep learning model, exploring techniques such as grid search, random search, and Bayesian optimization. We will analyze the computational requirements of different models and feature extraction techniques, with an eye towards potential real-time applications.

A comparative analysis between our deep learning approaches and traditional machine learning methods (e.g., SVM, KNN) will be conducted to understand the trade-offs between model complexity, performance, and generalizability. We will also critically assess the limitations of our approach and the dataset, proposing future research directions to address these limitations.

Throughout the project, we will maintain detailed documentation of all methodologies, algorithms, and experimental setups. The codebase will be prepared for open-source release to ensure reproducibility of our results and to contribute to the broader research community.

By systematically exploring various deep learning architectures and addressing the challenges of inter-subject variability, this project aims to make a significant contribution to the fields of affective computing and brain-computer interfaces. The ultimate goal is to push the boundaries of EEG-based emotion recognition, moving closer to the realization of subject-independent systems that can accurately interpret emotional states across diverse individuals.

PROJECT DESCRIPTION AND GOALS

2.1 Literature Review

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

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.

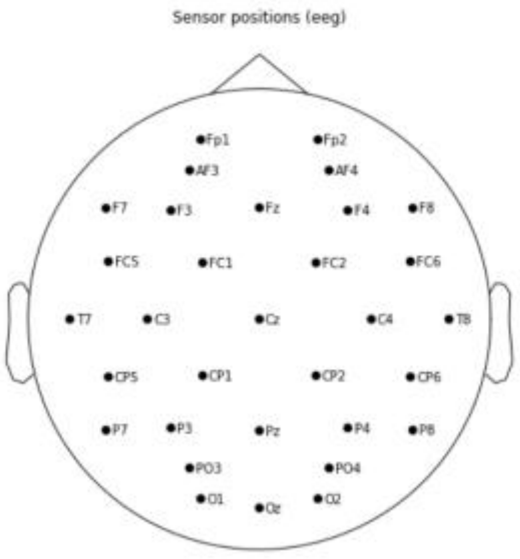


Figure 1. EEG electrodes position in DEAP dataset

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

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

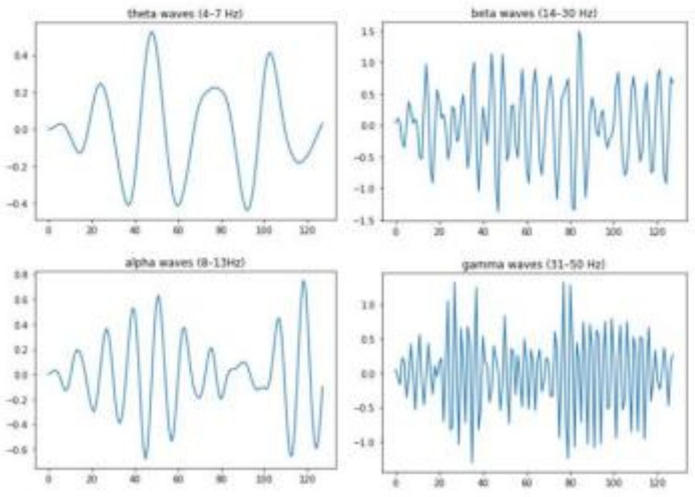


Figure 2. Four band wave using butter bandpass filter

Calculating power spectral density (PSD) 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' autocorrelation 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.

2.1.2 Continuous 3-dimensional emotion model

To quantitively measure human emotions, state of art 3-dimensional emotion model is widely used in research. Unlike the discrete emotion model, it offers higher resolution when characterizing ambiguous emotions . 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

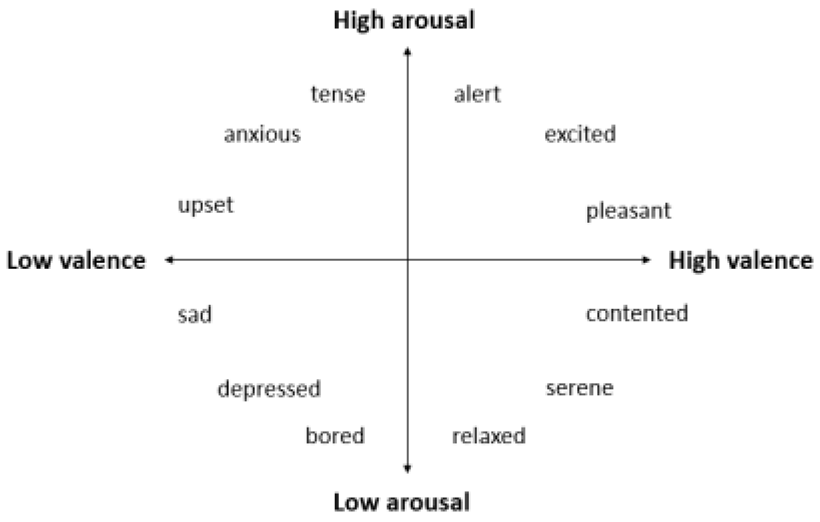


Figure 3. 2-dimensional emotion model

2.1.3 Review of EEG emotion dataset

2.1.3.1 DEAP

DEAP emotion dataset 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.

2.1.3.2 SEED

SEED 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.

2.1.4 Review of machine learning method

2.1.4.1 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. 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. 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 of its capability of finding the optimal non-linear hyperplane.Some papers 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.

2.1.5 Review of deep learning method

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

2.1.5.1 Convolutional Neural network

Some 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. Some papers 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.
Some papers 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, some 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

2.1.5.2 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.

In Google’s work, 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. 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. Some research papers innovatively proposed an end-toend 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.

2.1.5.3 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.

2.1.5.4 Autoencoder

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

2.1.5.5 Graph Neural network

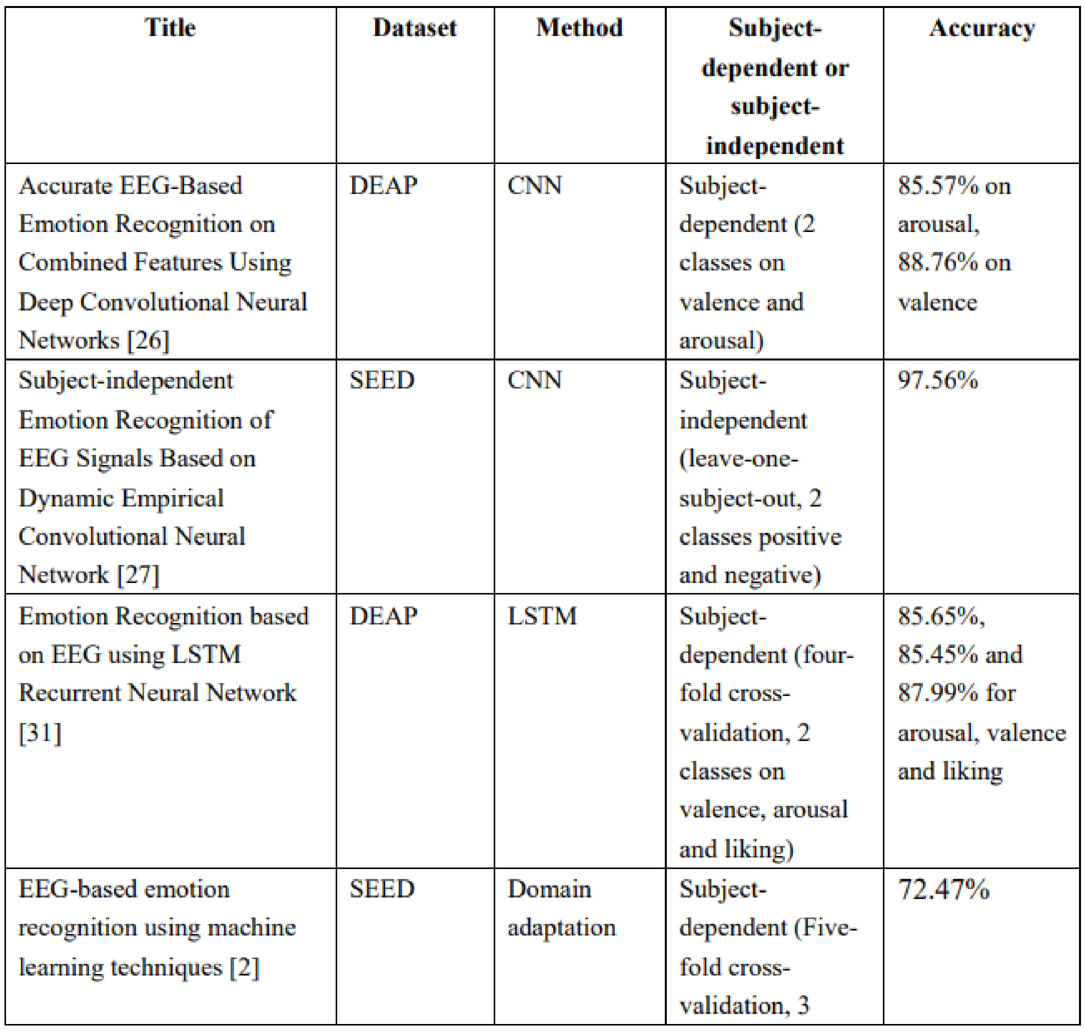
Recently, some other researchers are working on finding the relationship between different EEG channels using graph neural networks (GNN). proposed dynamical graph convolutional neural networks (DGCNN) which can effectively extract more
discriminative features between multiple channels. They improved the accuracy of subject-dependent experiments and subject-independent experiments up to 90.4% and
79.95% on SEED dataset. Inspired by many papers, 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.

2.1.5.6 Hybrid Neural network

Xin et al . 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) 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 unlabelled target domain. They improved the accuracy to 80.46% compared to their previous work.

Another hybrid deep learning model is proposed in . 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 modelling 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” , 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 27 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 crossvalidation 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%

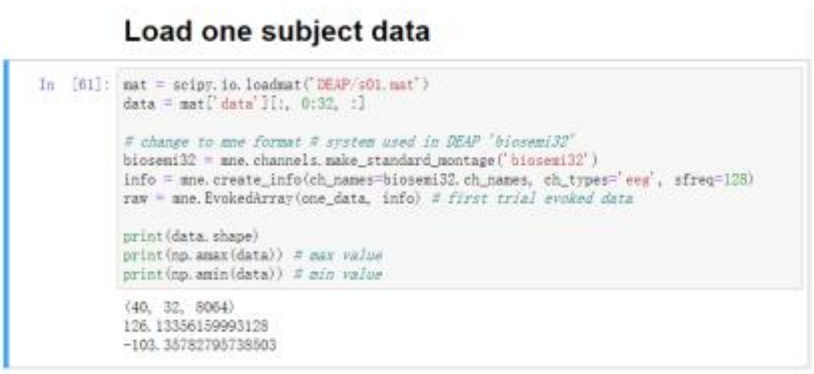
			Class: positive, neutral, negative)	
EEG-based emotion recognition using machine learning techniques [2]	SEED	Autoencoder	Subject-dependent (Five-fold cross-validation, 3 Class: positive, neutral, negative)	59.66%
Three class emotions recognition based on deep learning using staked autoencoder [34]	SEED	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

2.1.5.7 Review of Python package MNE

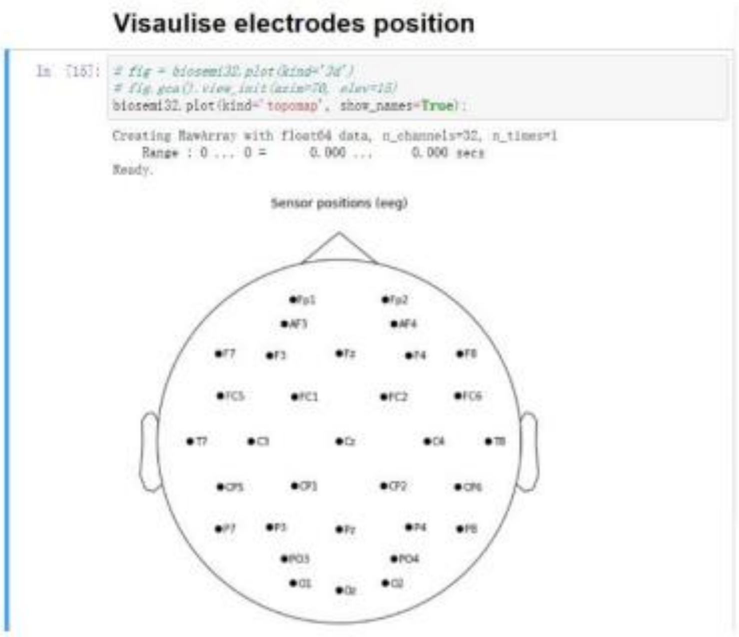
MNE is an open-source Python package for exploring, visualizing, and analyzing human neurophysiological data: MEG, EEG, ECoG, NIRS, and more [11]. It has a lot of useful functions like EEG pre-processing, EEG electrode visualization, artifact detection, Independent Components Analysis (ICA) and so on. 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
```

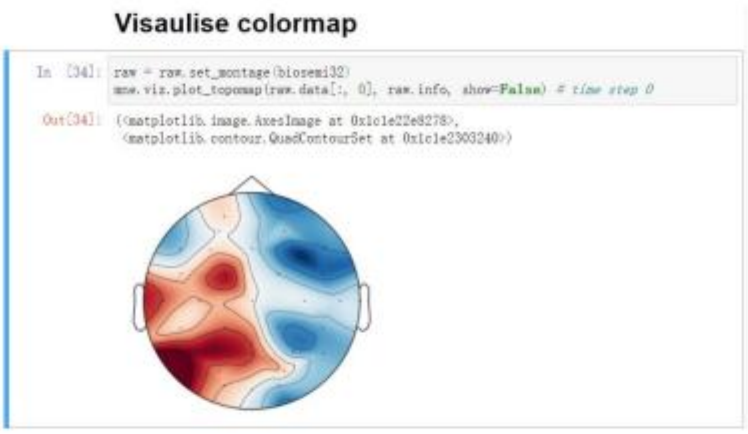
Figure 4. Code of creating a MNE raw object

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


Visaulise electrodes position

```
In [15]: # fig = biosemi32.plot(kind='3d')
         # fig.gca().view_init(azim=70, elev=15)
         biosemi32.plot(kind='topomap', show_names=True);

Creating RawArray with float64 data, n_channels=32, n_times=1
    Range : 0 ... 0 =      0.000 ...     0.000 secs
Ready.
```

Figure 5. EEG electrodes position in DEAP dataset

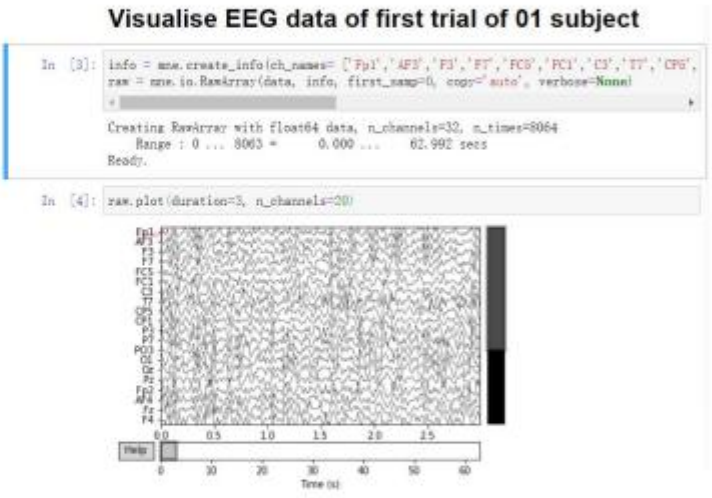

Visaulise colormap

```
In [34]: raw = raw.set_montage(biosemi32)
         mne.viz.plot_topomap(raw.data[:, 0], raw.info, show=False) # time step 0

Out[34]: (<matplotlib.image.AxesImage at 0x1c1e22e8278>,
          <matplotlib.contour.QuadContourSet at 0x1c1e2303240>)
```

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

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



Visualise EEG data of first trial of 01 subject

```
In [3]: info = mne.create_info(ch_names= ['Fp1','AF3','F3','F7','FC5','FC1','C3','T7','CP5',
        raw = mne.io.RawArray(data, info, first_samp=0, copy='auto', verbose=None)

        Creating RawArray with float64 data, n_channels=32, n_times=8064
            Range : 0 ... 8063 =      0.000 ...    62.992 secs
        Ready.

In [4]: raw.plot(duration=3, n_channels=20)
```

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.

2.2 Gaps Identified

This research project aims to advance the field of EEG-based emotion recognition by leveraging state-of-the-art deep learning techniques, with a particular focus on achieving subject-independent classification. The cornerstone of our study will be the DEAP (Database for Emotion Analysis using Physiological Signals) dataset, which provides a rich collection of EEG recordings from 32 subjects, each undergoing 40 trials. Our analysis will center on the valence and arousal dimensions of emotion, seeking to develop robust methods for binary classification in these emotional spaces.

The project will begin with a comprehensive examination of the DEAP dataset, including thorough preprocessing to clean and normalize the EEG signals. We will explore a wide array of feature extraction methods, spanning time-domain, frequency-domain, and time-frequency domain approaches. Special attention will be given to Power Spectral Density (PSD) analysis for deriving Frequency Band Power (FBP) features across the standard EEG frequency bands: Delta, Theta, Alpha, Beta, and Gamma. These features will serve as the foundation for our deep learning models.

At the heart of our research will be the development and comparison of several deep learning architectures. We will implement Convolutional Neural Networks (CNNs) to capture spatial features within the EEG signals, Long Short-Term Memory (LSTM) networks to model temporal dependencies, and Dense Neural Networks (DNNs) for high-level feature learning. A key focus will be on hybrid models, particularly the SAE+LSTM (Stacked Autoencoder + LSTM) architecture, which has shown promise in recent literature. Additionally, we will explore the potential of Graph Neural Networks (GNNs) to model the complex inter-channel relationships present in EEG data.

The primary challenge and objective of this project is to achieve subject-independent classification. To this end, we will implement and evaluate various domain adaptation techniques and transfer learning approaches. These methods will be crucial in addressing the significant inter-subject variability inherent in EEG data and moving towards a more generalizable emotion recognition system.

Our evaluation methodology will employ 10-fold cross-validation to ensure robust performance assessment. We will focus on binary classification for both valence and arousal dimensions separately, utilizing standard performance metrics such as accuracy, precision, recall, and F1-score. A comprehensive comparison with state-of-the-art methods reported in the literature will be conducted to contextualize our results within the field.

Visualization and interpretation of our models will be a key component of the project. We will leverage the MNE Python package for EEG signal visualization and develop techniques to interpret the learned features and decision-making processes of our deep learning models. This will not only aid in understanding the relationship between EEG patterns and emotional states but also provide insights that could guide future research in the field.

The project will also involve systematic hyperparameter optimization for each deep learning model, exploring techniques such as grid search, random search, and Bayesian optimization. We will analyze the computational requirements of different models and feature extraction techniques, with an eye towards potential real-time applications.

A comparative analysis between our deep learning approaches and traditional machine learning methods (e.g., SVM, KNN) will be conducted to understand the trade-offs between model complexity, performance, and generalizability. We will also critically assess the limitations of our approach and the dataset, proposing future research directions to address these limitations.

Throughout the project, we will maintain detailed documentation of all methodologies, algorithms, and experimental setups. The codebase will be prepared for open-source release to ensure reproducibility of our results and to contribute to the broader research community.

By systematically exploring various deep learning architectures and addressing the challenges of inter-subject variability, this project aims to make a significant contribution to the fields of affective computing and brain-computer interfaces. The ultimate goal is to push the boundaries of EEG-based emotion recognition, moving closer to the realization of subject-independent systems that can accurately interpret emotional states across diverse individuals.

2.3. Objectives

2.3.1 EEG Signal Data Preprocessing

The first objective of the project involves the preprocessing of EEG signals to standardize and normalize the data into a suitable format. This step is crucial as it prepares the raw EEG data for further analysis by removing artifacts, noise, and inconsistencies that could adversely affect the model's performance. Standardization ensures that the data has a mean of zero and a standard deviation of one, while normalization rescales the data to a specific range, typically between 0 and 1. These preprocessing techniques enhance the quality of the input data, facilitating more accurate feature extraction and model training.

2.3.2 Feature Extraction

Following preprocessing, the next objective is to extract relevant features from the EEG signals using Welch's method to compute Power Spectral Density (PSD) and derive Frequency Band Power (FBP) features. Welch's method involves dividing the EEG signal into overlapping segments, calculating a modified periodogram for each segment, and averaging these periodograms to obtain a smooth estimate of the power spectrum. This technique effectively reduces noise in the power spectra, allowing for a clearer representation of frequency components. The extracted FBP features serve as critical inputs for subsequent modeling steps, capturing essential characteristics of brain activity associated with different emotional states.

2.3.3 Design of the Stacked Autoencoder

The design of a Stacked Autoencoder (SAE) is aimed at reducing dimensionality and extracting high-level features from the pre-processed EEG data. The SAE consists of multiple layers where each layer learns to encode input data into a lower-dimensional representation before reconstructing it back to its original form. This architecture not only compresses the data but also helps in identifying complex patterns within the EEG signals that are relevant for emotion recognition. By leveraging this approach, the model can focus on significant features while discarding irrelevant noise, thus enhancing its predictive capabilities.

2.3.4 Implementation of the Long Short-Term Memory Network

The implementation of a Long Short-Term Memory (LSTM) network is crucial for modeling temporal dependencies in the feature sequences derived from the SAE. LSTMs are designed to capture long-range dependencies in sequential data, making them particularly effective for time-series analysis such as EEG signals. By processing sequences of features over time, LSTMs can learn how emotional states evolve, allowing for more accurate predictions based on historical context. This capability is essential for recognizing emotions that may manifest over extended periods rather than at discrete moments.

2.3.5 Training through a Mini-Batch Gradient Descent Optimizer

Training through a mini-batch gradient descent optimizer is employed to optimize both the SAE and LSTM models efficiently. This technique divides the training dataset into smaller batches, allowing for faster convergence and reduced computational load during training. By updating model parameters based on mini-batches rather than the entire dataset at once, this approach enhances training speed while maintaining stability in learning. The use of mini-batch gradient descent is particularly beneficial in deep learning applications where large datasets are common.

2.3.6 Testing and Comparison of Model Performance Parameters

Finally, testing and comparison of model performance parameters involve evaluating accuracy in classifying emotions based on valence and arousal dimensions while comparing results against baseline methods like Support Vector Machine (SVM). Performance metrics such as accuracy, precision, recall, and F1-score are computed to assess how well the model performs in distinguishing between different emotional states. By benchmarking against traditional methods like SVM, which has been widely used in emotion recognition tasks, this objective provides insights into the effectiveness of deep learning approaches like SAE+LSTM in achieving superior classification performance.

2.4 Problem Statement

The increasing demand for effective emotion recognition systems in various applications, such as healthcare, gaming, and human-computer interaction, has highlighted the limitations of traditional emotion recognition methods that rely on facial expressions or auditory cues. These methods often fail in scenarios where visual or audio data is compromised, such as low-light environments or when users do not exhibit observable emotional expressions. In contrast, electroencephalogram (EEG) signals provide a direct measure of brain activity and have shown promise in accurately capturing emotional states. However, existing EEG-based emotion recognition systems primarily operate on a subject-dependent basis, which limits their generalizability across different individuals.

This project aims to develop a novel deep learning architecture that combines Stacked Autoencoders (SAE) and Long Short-Term Memory (LSTM) networks to achieve subject-independent emotion classification from EEG data. By leveraging advanced feature extraction techniques and robust validation methods, the project seeks to enhance the accuracy and reliability of emotion recognition systems.

2.5 Project Plan

Research and Literature Review

- Conduct an extensive literature review on existing EEG-based emotion recognition methodologies.
- Identify gaps in current research and establish the significance of using deep learning techniques.

Dataset Preprocessing

- Obtain the DEAP dataset containing EEG recordings.
- Implement data preprocessing steps including standardization, normalization, and noise removal.

Feature Extraction

- Apply Welch's method to calculate Power Spectral Density (PSD) and extract Frequency Band Power (FBP) features from the pre-processed EEG signals.

Model Development

- Design and implement the Stacked Autoencoder to reduce dimensionality and extract high-level features.
- Develop the Long Short-Term Memory (LSTM) network to model temporal dependencies in the feature sequences.

Training

- Train the SAE and LSTM models using a mini-batch gradient descent optimizer with Mean Squared Error (MSE) as the loss function.
- Optimize hyperparameters based on validation performance.

Validation and Testing

- Conduct 10-fold cross-validation to evaluate model performance across different subsets of data.
- Test the model's accuracy in classifying emotions based on valence and arousal dimensions.

Results Analysis and Comparison

- Analyse results using performance metrics such as accuracy.
- Compare results against baseline models like Support Vector Machine (SVM).

Report Making

- Compile a comprehensive report detailing methodologies, experiments conducted, results obtained, conclusions drawn, and recommendations for future work.

REQUIREMENT ANALYSIS

3.1. Functional Requirements

The functional requirements outline the specific tasks the system must perform. For data preprocessing, the system must implement data cleaning to remove artifacts, missing values, and noise from the raw EEG signals. This includes applying signal processing techniques such as bandpass and notch filters to eliminate unwanted frequencies and baseline drift. Additionally, the EEG signals need to be segmented into relevant time windows for analysis. In terms of feature extraction, the system should derive time-domain features like mean, variance, skewness, and kurtosis. It should also utilize frequency-domain techniques like Fast Fourier Transform (FFT) to compute power spectral density (PSD) from the EEG signals and analyze spatial relationships between EEG channels. The extracted features will then be mapped to emotion categories such as arousal, valence, and dominance.When it comes to model building, the system must facilitate model training and evaluation using both traditional machine learning algorithms like Support Vector Machines (SVM), Random Forests, and K-Nearest Neighbors (KNN), as well as deep learning models such as Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks for emotion classification. The model should employ cross-validation techniques, particularly k-fold cross-validation or leave-one-subject-out methods, to ensure generalization across different participants. Furthermore, it must handle multi-label classification by predicting multi-dimensional emotion labels rather than just binary classifications.For emotion prediction, the system should be capable of predicting the emotional state of a subject based on unseen EEG data while providing a confidence score for each predicted emotion class. The results analysis and visualization component should report performance metrics such as accuracy, precision, recall, and F1-score while visualizing results through confusion matrices and other graphical representations of EEG signals, extracted features, and model predictions.

3.2 Non-Functional Requirements

The non-functional requirements define the qualities or attributes of the system. In terms of performance, the system should be optimized for real-time or near-real-time emotion recognition during testing or deployment phases. Regarding scalability, it must accommodate larger EEG datasets or additional participants in future project extensions. Usability is another critical aspect; the system should provide an intuitive interface for users to interact with the emotion recognition process seamlessly. Reliability is essential to ensure consistent performance across various conditions and datasets. Lastly, given that EEG data represents sensitive biometric information, security measures such as data encryption and secure storage are crucial to protect participant privacy.

3.3 Hardware Requirements

The hardware requirements include an EEG acquisition device equipped with a sufficient number of channels (e.g., 32 or 64 electrodes) to capture brain activity necessary for emotion recognition at a high sampling rate (≥ 128 Hz). For computation hardware, a powerful GPU (e.g., NVIDIA RTX 3090) is necessary if deep learning models are employed to expedite model training and inference. At least 16 GB of RAM is recommended to handle memory requirements for large EEG datasets, along with sufficient storage space for the DEAP dataset, processed data, and model checkpoints. Peripheral devices such as monitors are required for visualizing real-time signals and training progress.

3.4. Software Requirements

On the software side, Python will serve as the primary programming language for data processing, machine learning, and deep learning tasks. Optional use of MATLAB may be considered for advanced signal processing tasks. The project will leverage various libraries and frameworks: TensorFlow/Keras or PyTorch for implementing deep learning models; Scikit-learn for traditional machine learning algorithms; MNE-Python for EEG data analysis; SciPy for advanced signal filtering; Pandas for data manipulation; NumPy for numerical computations; and Matplotlib/Seaborn or Plotly for visualizations. Integrated Development Environments (IDEs) like Jupyter Notebooks or PyCharm will facilitate coding and debugging processes on compatible operating systems such as Linux or Windows.

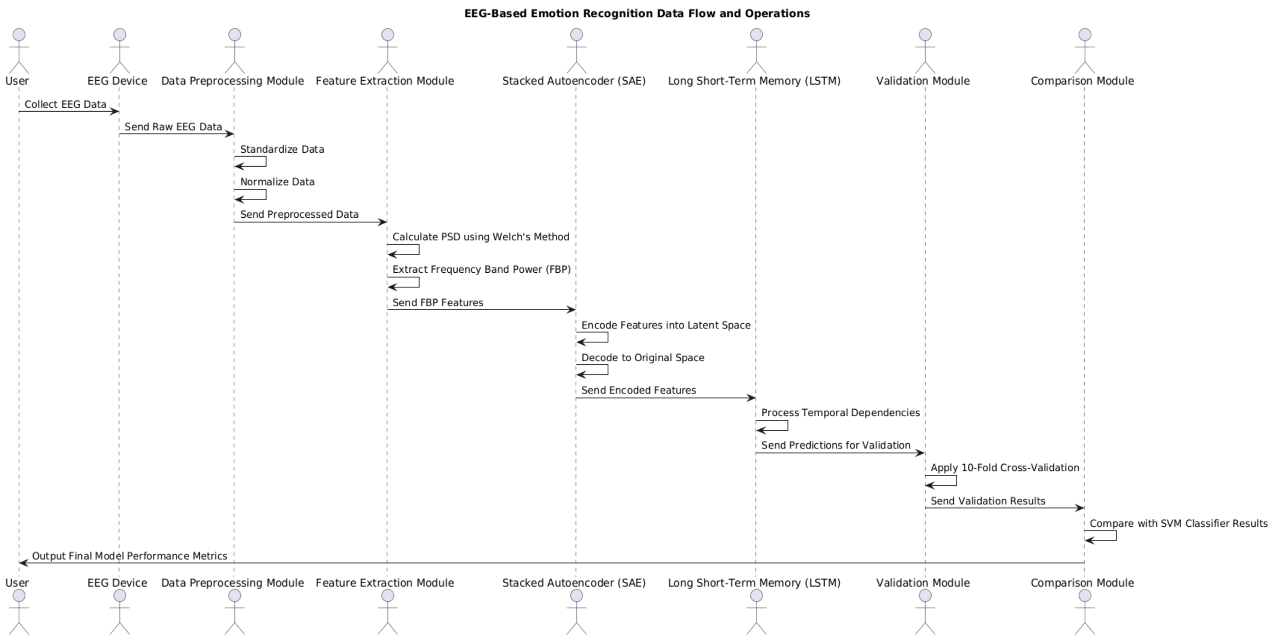
3.5. Data Requirements

The project will utilize raw EEG data from the DEAP dataset while ensuring proper preprocessing to remove noise and artifacts for reliable emotion recognition. Emotion labeling will involve using self-reported labels (arousal, valence, dominance) from the DEAP dataset to guide supervised learning processes. Given that the dataset contains recordings from only 32 participants, data augmentation techniques like time-shifting or adding Gaussian noise may be employed to increase training set size and improve generalization.

3.6. Constraints

Data constraints include ensuring that all preprocessing techniques effectively clean the EEG signals without losing critical information necessary for accurate emotion classification. The system must also accommodate any limitations posed by small sample sizes while maintaining high accuracy in predictions across diverse subjects.In summary, this comprehensive requirement analysis outlines the essential components needed for developing an effective EEG-based emotion recognition system that leverages advanced machine learning techniques while ensuring usability, security, and scalability in real-world applications.

SYSTEM DESIGN



EEG-Based Emotion Recognition Data Flow and Operations
User
EEG Device
Data Preprocessing Module
Feature Extraction Module
Stacked Autoencoder (SAE)
Long Short-Term Memory (LSTM)
Validation Module
Comparison Module
Collect EEG Data
Send Raw EEG Data
Standardize Data
Normalize Data
Send Preprocessed Data
Calculate PSD using Welch's Method
Extract Frequency Band Power (FBP)
Send FBP Features
Encode Features into Latent Space
Decode to Original Space
Send Encoded Features
Process Temporal Dependencies
Send Predictions for Validation
Apply 10-Fold Cross-Validation
Send Validation Results
Compare with SVM Classifier Results
Output Final Model Performance Metrics

Figure

The system begins with the User, who initiates the process by collecting EEG data from an EEG Device. This data is then transmitted to the Data Preprocessing Module, where it undergoes essential preprocessing steps to prepare it for further analysis. This module performs standardization and normalization of the raw EEG signals, ensuring that they are in a suitable format for subsequent processing.

Once the data is preprocessed, it is sent to the Feature Extraction Module. Here, Welch's method is applied to calculate the Power Spectral Density (PSD) of the EEG signals. This step is crucial as it allows for the extraction of Frequency Band Power (FBP) features, which are essential for identifying emotional states based on brain activity. The extracted features are then forwarded to the Stacked Autoencoder (SAE).

The Stacked Autoencoder plays a pivotal role in encoding the FBP features into a lower-dimensional latent space, effectively reducing dimensionality while preserving critical information. After encoding, these features are reconstructed back to their original form, demonstrating the autoencoder's ability to learn meaningful representations from the input data. The encoded features are then passed to the Long Short-Term Memory (LSTM) network, which processes these sequences to capture temporal dependencies inherent in EEG data.During this phase, training occurs through a Mini-Batch Gradient Descent optimizer, utilizing Mean Squared Error (MSE) as the loss function. This optimization process ensures that the LSTM model learns effectively from the training data.

Following training, the model's performance is validated using a 10-fold cross-validation. This validation technique assesses how well the model generalizes across different subsets of data, ensuring robustness in its predictions. The results from this validation phase are then forwarded to a Comparison Module, where they are compared against baseline models such as Support Vector Machine (SVM) classifiers.

Finally, after processing through these modules, performance metrics are calculated and compared with traditionally implemented methods to accurately undertstand the relative performance of the model

4.1 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), $s1$, $s2$, …, $s12$ are the source signal from the 12 different brain regions and $a1$, $a2$, …, $a12$ are corresponding coefficients.

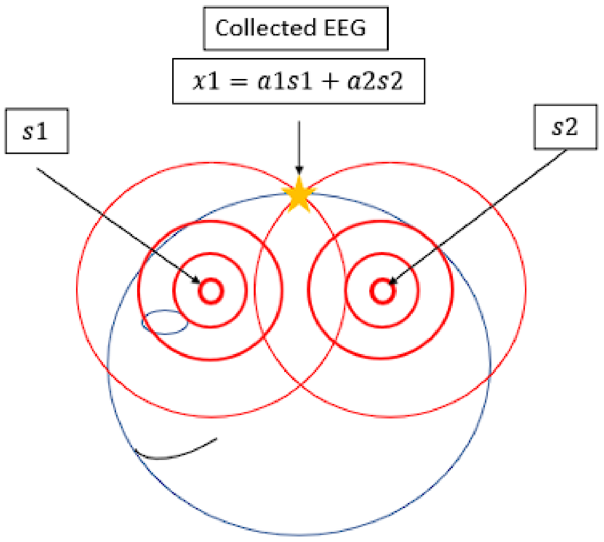


Figure 9. Linear EEG signal mixing model

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.

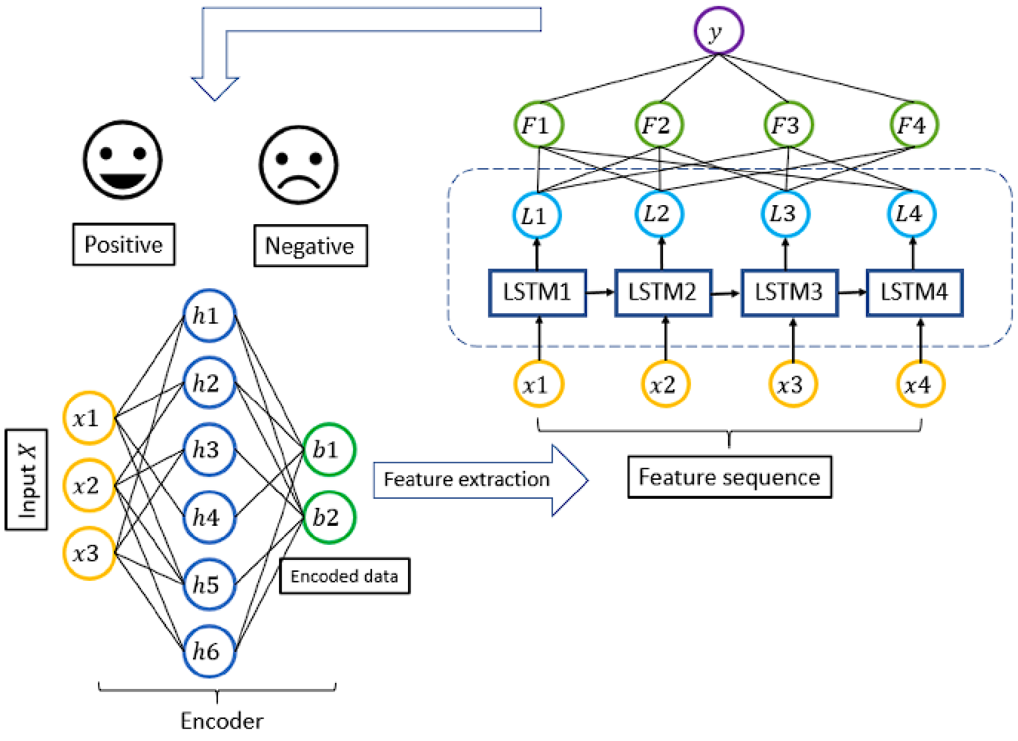
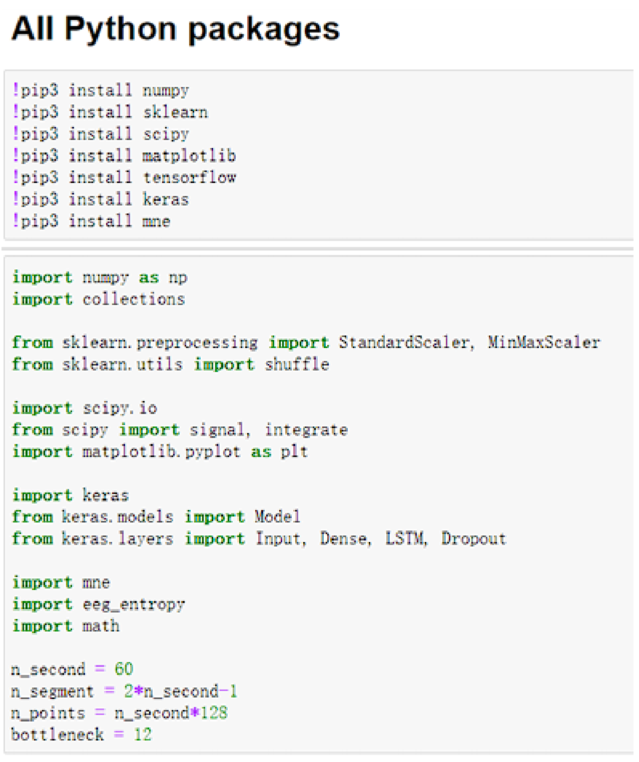


Figure 10. Structure of SAE+LSTM algorithm

4.2 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

4.3 EEG Data Processing

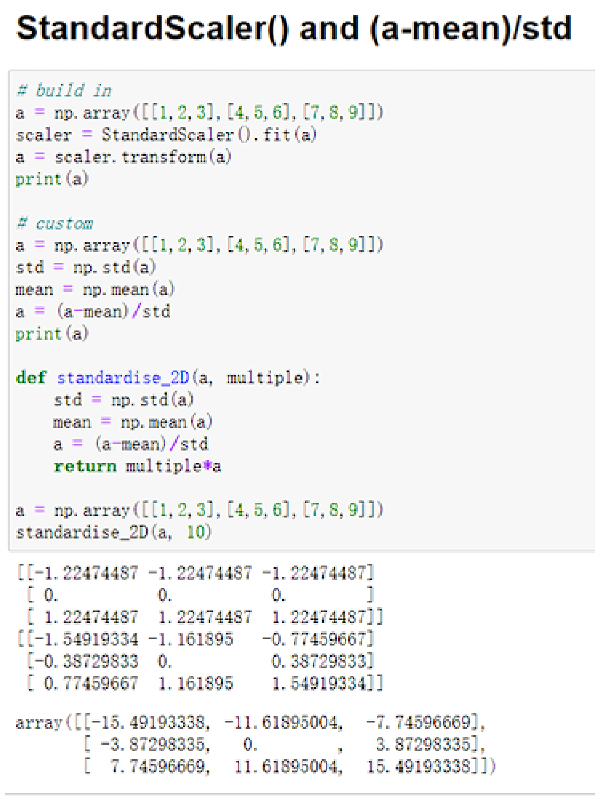
Deep learning engineers realized that data scaling is a crucial step to conduct because potential gradient exploding or gradient vanishing may severely decrease the accuracy of classification. The two most commonly used effective data scaling techniques are data standardization and data normalization. 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.

4.3.1 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

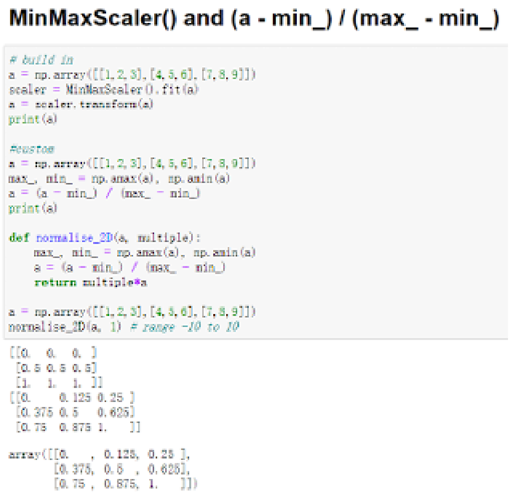
To use the built-in function, we can use “from sklearn.preprocessing import StandardScaler, MinMaxScaler” to import two built-in functions. But as shown in Figure 12, the default StandardScaler can only scale data on every column but not the whole matrix. So we need to first find the mean and standardbred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

4.3.2 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 $xnorm$ is normalized data, $xmin$ is the minimum value of all data and $xmax$ 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
```

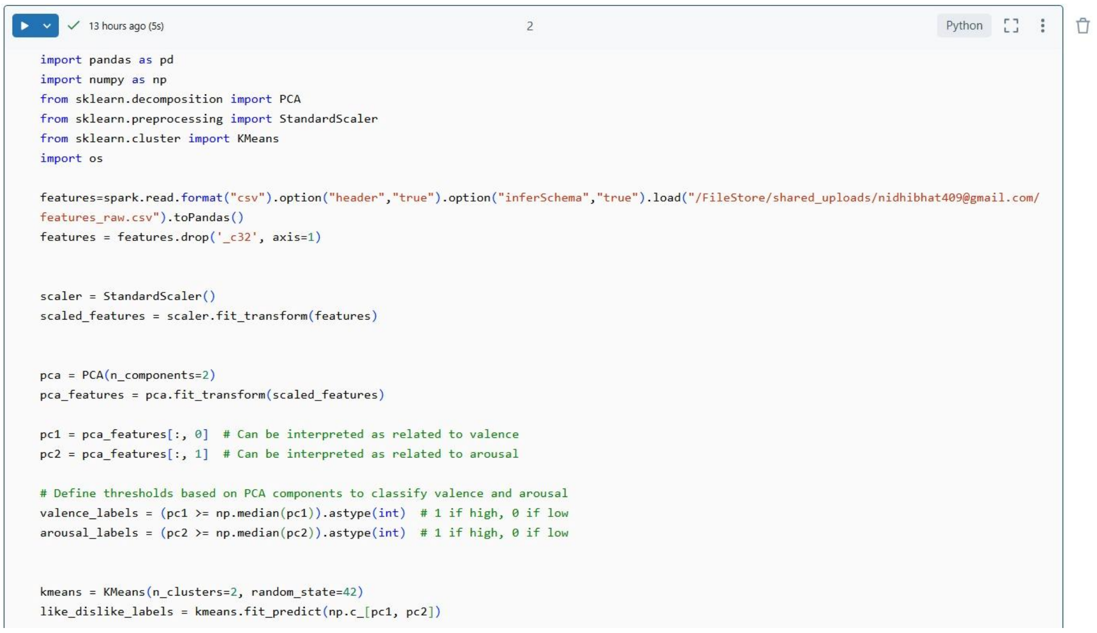
```
[[0.  0.  0. ]
 [0.5 0.5 0.5]
 [1.  1.  1. ]]
[[0.    0.125 0.25 ]
 [0.375 0.5   0.625]
 [0.75  0.875 1.   ]]

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

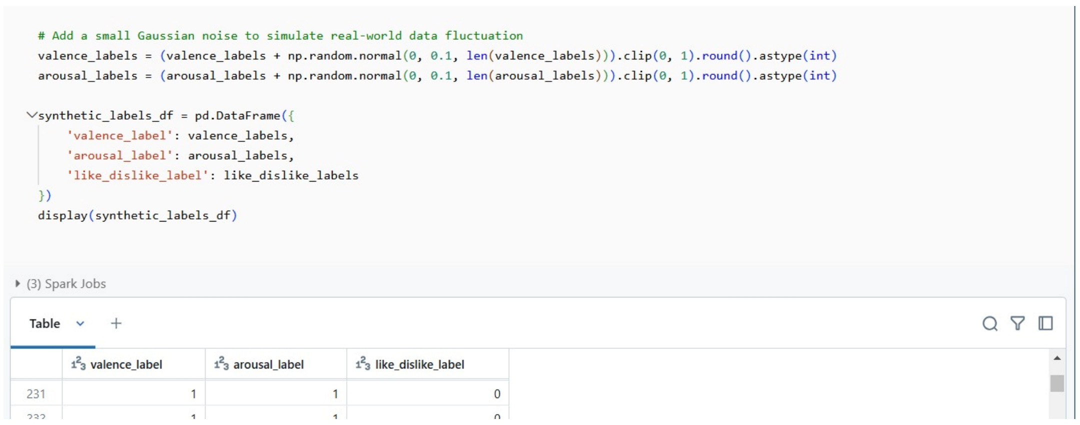
To use the built-in function, we can use “from sklearn.preprocessing import StandardScaler, MinMaxScaler” to import two built-in functions. But as shown in Figure 12, the default StandardScaler can only scale data on every column but not the whole matrix. So we need to first find the mean and standardbred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

4.4 Synthetic Label Generation

To generate synthetic labels for the DEAP dataset- due to the incomplete datset issues we were facing, we first standardized the features using StandardScaler to ensure consistent scaling, reducing bias and enhancing the performance of subsequent analyses. We applied Principal Component Analysis (PCA) to extract the first two principal components (PC1 and PC2), which captured the primary variance in the data. These components were used as proxies for emotional states: PC1 was hypothesized to correlate with valence (positivity or negativity), and PC2 with arousal (activation level). We created binary labels by comparing PC1 and PC2 to their median values, assigning 1 for higher scores and 0 otherwise. Additionally, K-Means clustering on PC1 and PC2 simulated a preference classification. To mimic the natural variability found in physiological signals and to reduce overfitting, Gaussian noise was added to the valence and arousal labels. The final dataset included three target variables—valence_label, arousal_label—which were used in subsequent machine learning experiments to predict emotional states.



```python
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import os

features=spark.read.format("csv").option("header","true").option("inferSchema","true").load("/FileStore/shared_uploads/nidhibhat409@gmail.com/
features_raw.csv").toPandas()
features = features.drop('_c32', axis=1)


scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)


pca = PCA(n_components=2)
pca_features = pca.fit_transform(scaled_features)

pc1 = pca_features[:, 0]  # Can be interpreted as related to valence
pc2 = pca_features[:, 1]  # Can be interpreted as related to arousal

# Define thresholds based on PCA components to classify valence and arousal
valence_labels = (pc1 >= np.median(pc1)).astype(int)  # 1 if high, 0 if low
arousal_labels = (pc2 >= np.median(pc2)).astype(int)  # 1 if high, 0 if low


kmeans = KMeans(n_clusters=2, random_state=42)
like_dislike_labels = kmeans.fit_predict(np.c_[pc1, pc2])
```



```python
# Add a small Gaussian noise to simulate real-world data fluctuation
valence_labels = (valence_labels + np.random.normal(0, 0.1, len(valence_labels))).clip(0, 1).round().astype(int)
arousal_labels = (arousal_labels + np.random.normal(0, 0.1, len(arousal_labels))).clip(0, 1).round().astype(int)

synthetic_labels_df = pd.DataFrame({
    'valence_label': valence_labels,
    'arousal_label': arousal_labels,
    'like_dislike_label': like_dislike_labels
})
display(synthetic_labels_df)
```

	valence_label	arousal_label	like_dislike_label
231	1	1	0
232	1	1	0

Code Snippet

4.5 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.

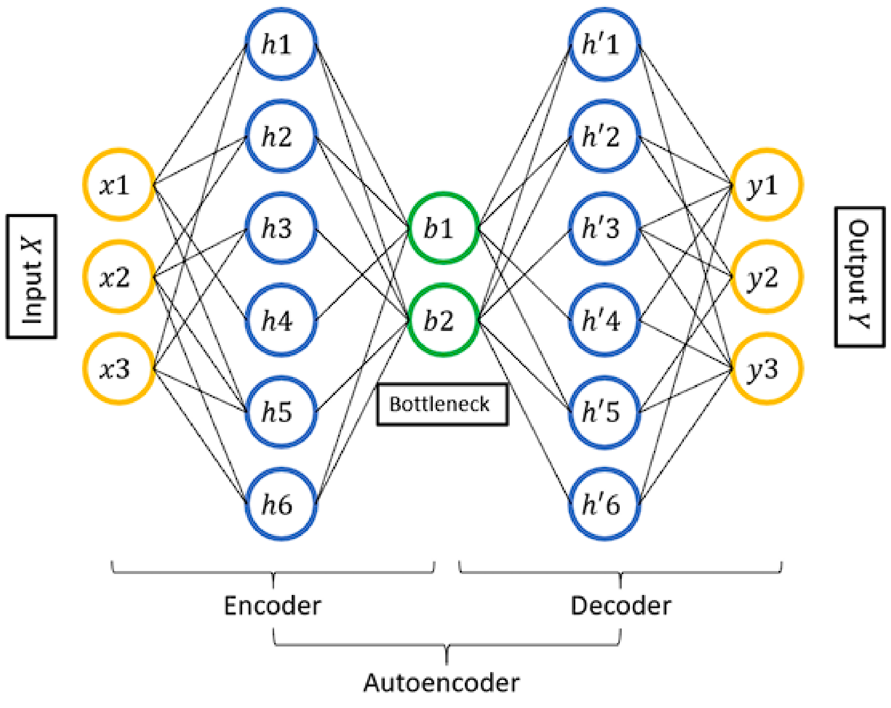


Figure 17. Illustrated stacked autoencoder structure

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.

```
input_dim = X_train.shape[1]
input_layer = Input(shape=(input_dim,))
encoder = Dense(128, activation='relu')(input_layer)
encoder = Dense(64, activation='relu')(encoder)
encoder_output = Dense(32, activation='relu')(encoder)

decoder = Dense(64, activation='relu')(encoder_output)
decoder = Dense(128, activation='relu')(decoder)
decoder_output = Dense(input_dim, activation='sigmoid')(decoder)

autoencoder = Model(input_layer, decoder_output)
autoencoder.compile(optimizer='adam', loss='mse')


autoencoder.fit(X_train, X_train, epochs=10, batch_size=32, shuffle=True, validation_data=(X_test, X_test))


encoder_model = Model(inputs=input_layer, outputs=encoder_output)
X_train_encoded = encoder_model.predict(X_train)
X_test_encoded = encoder_model.predict(X_test)


y_train = y_train.to_numpy().astype(np.float32).reshape(-1, 1)
y_test = y_test.to_numpy().astype(np.float32).reshape(-1, 1)


scaler_encoded = StandardScaler()
X_train_encoded = scaler_encoded.fit_transform(X_train_encoded)
X_test_encoded = scaler_encoded.transform(X_test_encoded)
```

After training the autoencoder model, we leverage its encoder part to transform the input EEG data into a lower-dimensional encoded representation. Prior to this step, we extract Power Spectral Density (PSD) features using Welch's method, which helps capture the frequency characteristics of the EEG signals, providing a robust representation of the underlying neural activity.

The extracted PSD features from each trial are then standardized and reshaped into a 3D tensor format required by the LSTM network, structured as (samples, timesteps, features), where:

- Samples denote the number of EEG trials.
- Timesteps correspond to the sequence length of the time steps (adjusted to fit the LSTM input).
- Features represent the encoded and standardized PSD features for each trial.

This transformation is crucial as LSTM networks are specifically designed to handle sequential data, allowing the model to capture temporal dependencies within the EEG signals. By converting the PSD features into this sequential format, the LSTM can effectively learn patterns and trends over time, enhancing its ability to predict target labels such as valence or arousal states.

```
y_train = y_train.to_numpy().astype(np.float32).reshape(-1, 1)
y_test = y_test.to_numpy().astype(np.float32).reshape(-1, 1)


scaler_encoded = StandardScaler()
X_train_encoded = scaler_encoded.fit_transform(X_train_encoded)
X_test_encoded = scaler_encoded.transform(X_test_encoded)


X_train_encoded = X_train_encoded.reshape(-1, 1, X_train_encoded.shape[1])
X_test_encoded = X_test_encoded.reshape(-1, 1, X_test_encoded.shape[1])
```

Optimization and Loss Function in Autoencoder: In our implementation, the autoencoder utilizes the Adam optimizer instead of Stochastic Gradient Descent (SGD). The Adam optimizer is chosen due to its adaptive learning rate, which tends to perform well in practice, especially for complex, non-convex problems like neural network training. The loss function employed is Mean Squared Error (MSE), which measures the reconstruction error by calculating the squared difference between the input features and their reconstructed outputs. The goal is to minimize this reconstruction error, effectively training the model to learn a compressed representation of the input data that captures its essential characteristics.

In this autoencoder setup, the training data is fed as both the input and the target output, as the autoencoder's objective is to reconstruct the input data as closely as possible. The compressed representation generated in the bottleneck layer (the encoded layer) captures the most significant and essential features of the original input data, making it a suitable input for subsequent classification tasks.

4.6 Result validation method

In our experiment, we utilize an 80-20 train-test split method for result validation instead of the traditional 10-fold cross-validation approach. Specifically, 80% of the total EEG trials are randomly selected as the training set, and the remaining 20% are used as the test set. This split allows us to train the model on a significant portion of the data while reserving a subset for evaluating its performance.

To ensure robustness and mitigate any potential bias due to the random splitting of data, we repeat this process with a fixed random seed (random_state=42). The training set is used to fit the models (autoencoder and LSTM), while the test set is reserved for evaluating the model's predictive performance using metrics such as accuracy and the confusion matrix. This method provides a consistent measure of model performance and helps in understanding its generalization capability on unseen data.

```
print("Feature shape before PSD extraction:", scaled_features.shape)

# Extract PSD features
X_psd = extract_psd_features(scaled_features)

# Print the shape of the extracted PSD features
print("Extracted PSD feature shape:", X_psd.shape)

y = labels['valence_label']

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_psd, y, test_size=0.2, random_state=42)


scaler_psd = StandardScaler()
X_train = scaler_psd.fit_transform(X_train)
X_test = scaler_psd.transform(X_test)
```

4.7 Feature Extraction Method

In our experiment, we utilize Welch’s method to extract Power Spectral Density (PSD) features from the EEG signals. Welch’s method is a widely used technique for estimating the power of a signal at different frequency components by dividing the signal into overlapping segments, applying a window function, computing the periodogram for each segment, and then averaging these periodograms.

We set the segment length (nperseg) to 128 data points, corresponding to a 1-second window for a sampling frequency of 128 Hz. No overlap is specified in the code, resulting in non-overlapping segments. The Hanning window is used to minimize spectral leakage when calculating the periodograms.

The PSD features are computed for each EEG channel using the welch() function from the Scipy library. The parameters specified include:

- Input EEG signal array: Standardized EEG data.
- Sampling rate (fs): Set to 128 Hz.
- Segment length (nperseg): Set to 128 points.
- Window type: Hanning window.

The extracted PSD values represent the power distribution of the EEG signal across different frequency components. These PSD features are flattened for each trial, forming a feature vector that serves as the input for the subsequent autoencoder and LSTM models.

This process captures the spectral characteristics of the EEG signals, enabling effective feature extraction and dimensionality reduction before feeding into the deep learning models.

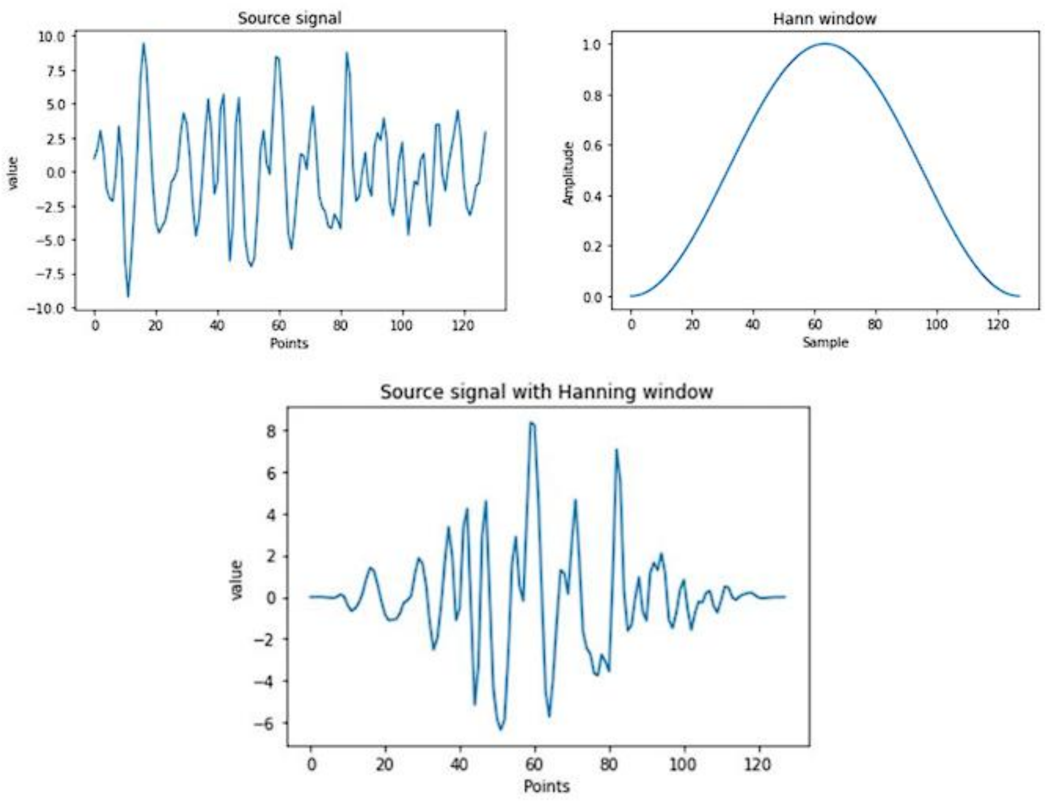
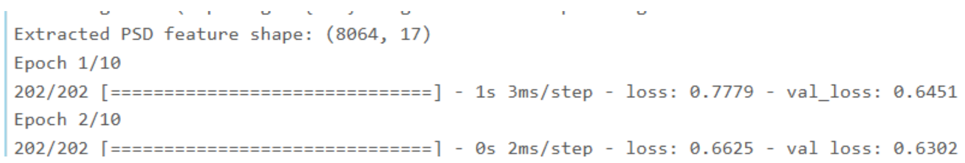


Figure 23. Illustration of Hanning window method



```
Extracted PSD feature shape: (8064, 17)
Epoch 1/10
202/202 [==============================] - 1s 3ms/step - loss: 0.7779 - val_loss: 0.6451
Epoch 2/10
202/202 [==============================] - 0s 2ms/step - loss: 0.6625 - val_loss: 0.6302
```

4.8 LSTM-RNN

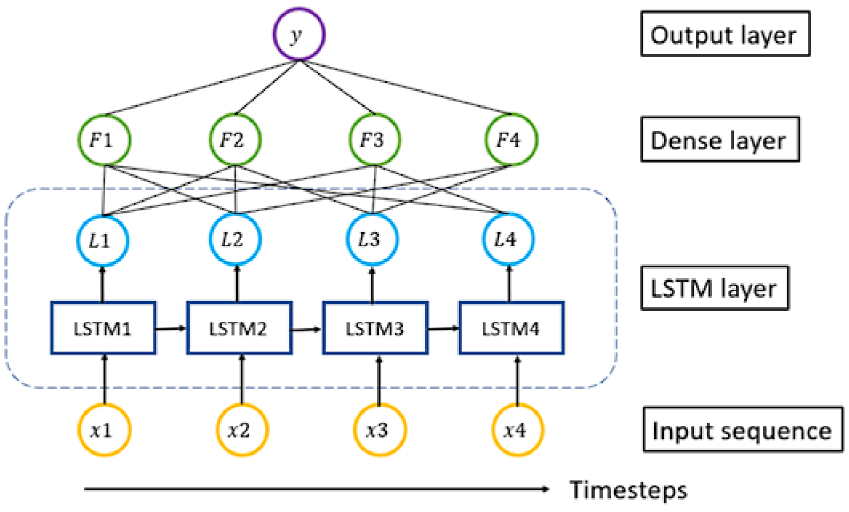
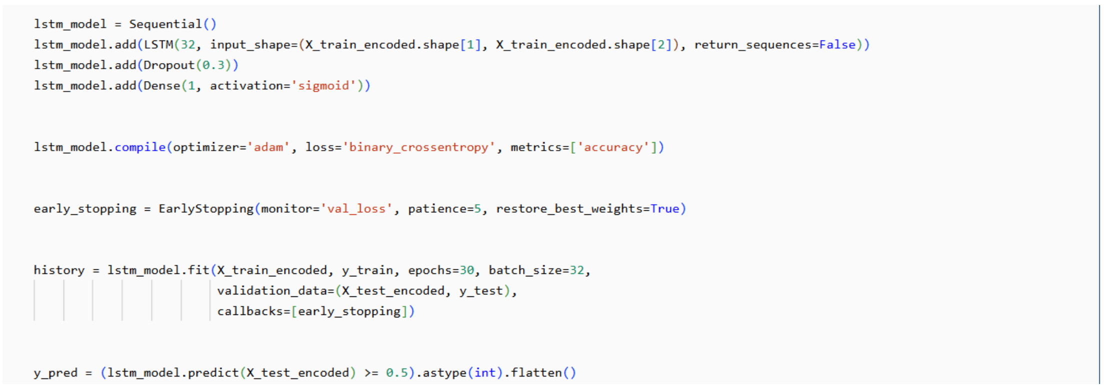


Figure 26. Structure of LSTM and dense layer

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.

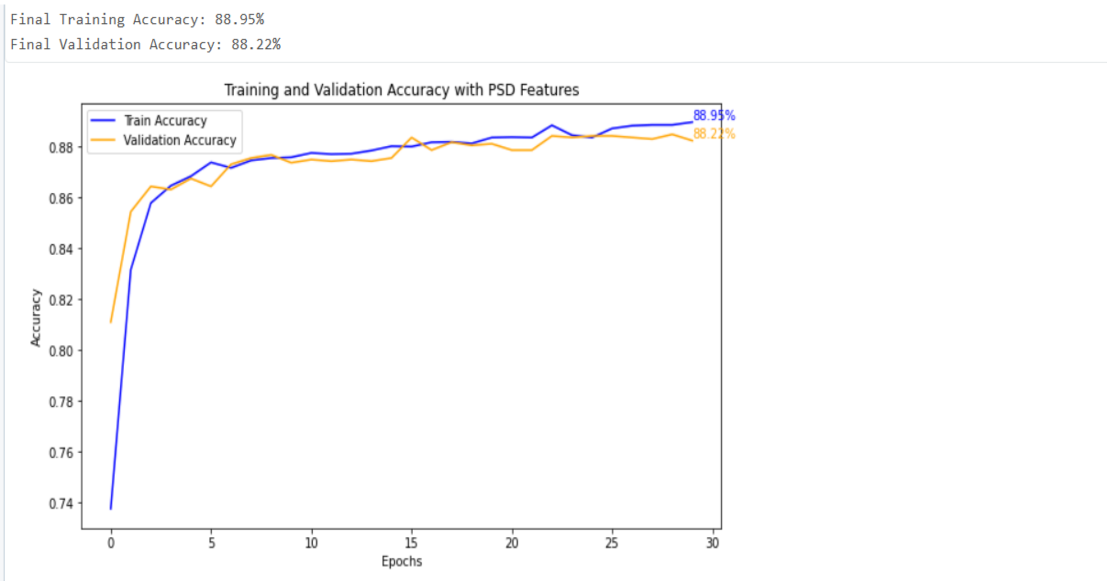


```
lstm_model = Sequential()
lstm_model.add(LSTM(32, input_shape=(X_train_encoded.shape[1], X_train_encoded.shape[2]), return_sequences=False))
lstm_model.add(Dropout(0.3))
lstm_model.add(Dense(1, activation='sigmoid'))


lstm_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])


early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)


history = lstm_model.fit(X_train_encoded, y_train, epochs=30, batch_size=32,
                         validation_data=(X_test_encoded, y_test),
                         callbacks=[early_stopping])


y_pred = (lstm_model.predict(X_test_encoded) >= 0.5).astype(int).flatten()
```

.

4.9 Results

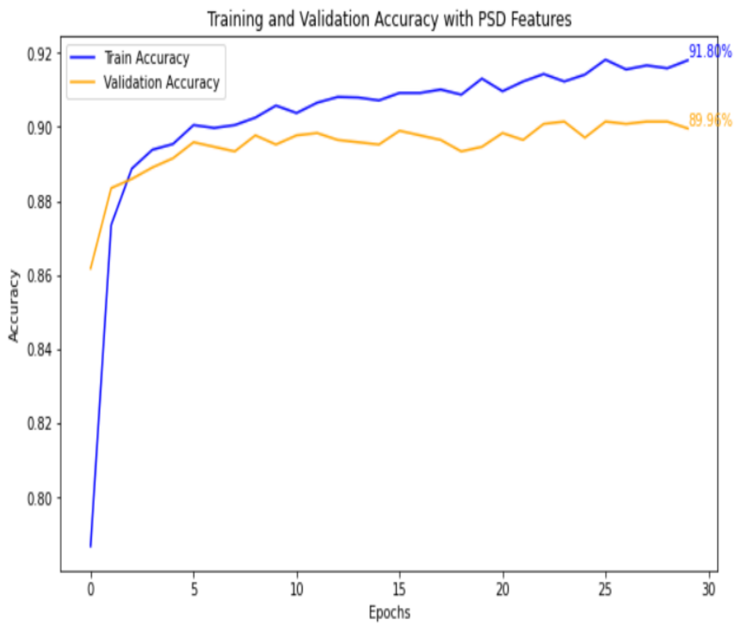
The results of the Autoencoder (AE) + LSTM method using a train-test split approach on the valence dimension are as follows. The model achieved an average test accuracy of 88.2% (for valence) 89.6% (for arousal). This evaluation was performed using 80% of the data for training and 20% for testing, with early stopping implemented to prevent overfitting. The confusion matrix analysis and accuracy plots further illustrate the model's performance across the training and validation phases.



```
Final Training Accuracy: 88.95%
Final Validation Accuracy: 88.22%
```

Valence

```
Final Training Accuracy: 91.80%
Final Validation Accuracy: 89.96%
```



Arousal

Conclusions and Future Work

5.1 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 AE+ 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 AE + 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.

5.2 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 59 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.

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 Report, 2018.

[3] F. Noroozi, M. Marjanovic, A. Njegus, S. Escalera, and G. Anbarjafari, "AudioVisual 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 Project No: A3272-201 65 [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 66 Project No: A3272-201 Learning System," in 2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), 3-6 Dec. 2018 2018, pp. 1240-1244, doi:

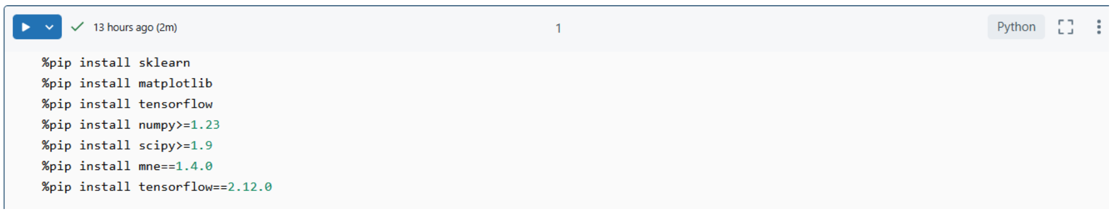
[37] X. Chai, Q. Wang, Y. Zhao, X. Liu, O. Bai, and Y. Li, "Unsupervised domain adaptation techniques based on auto-encoder for non-stationary EEG-based emotion recognition," Computers in Biology and Medicine, vol. 79, pp. 205-214, 2016/12/01/ 2016, doi: https://doi.org/10.1016/j.compbiomed.2016.10.019.

[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.

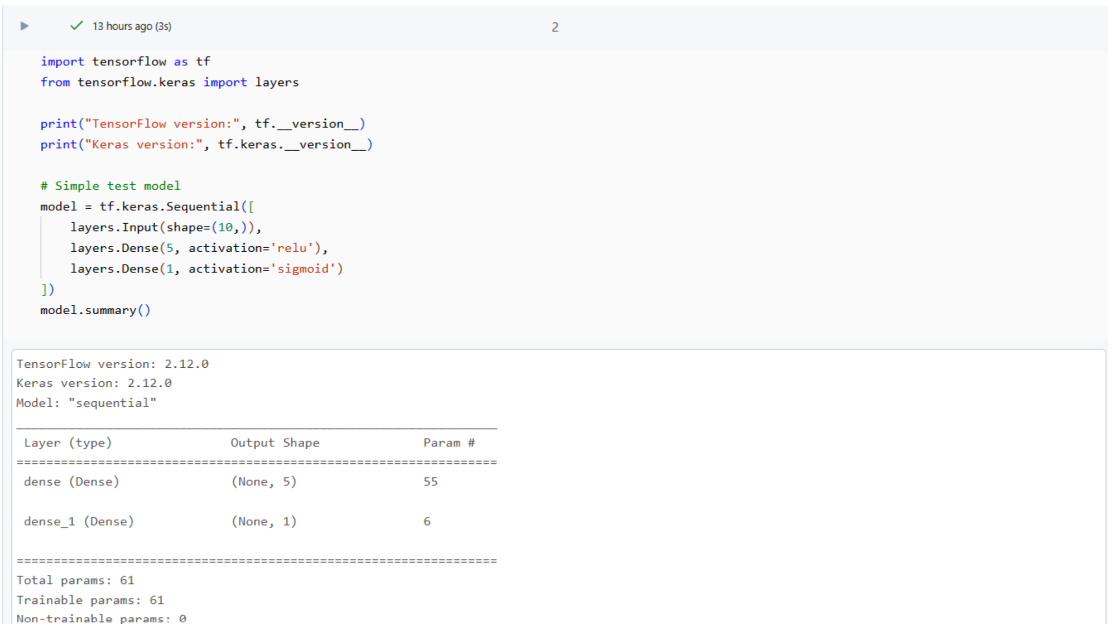
[39] Y. Li, J. Huang, H. Zhou, and N. Zhong, "Human Emotion Recognition with Electroencephalographic Multidimensional Features by Hybrid Deep Neural Networks," Applied Sciences, vol. 7, no. 10, 2017, doi: 10.3390/app7101060.

[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

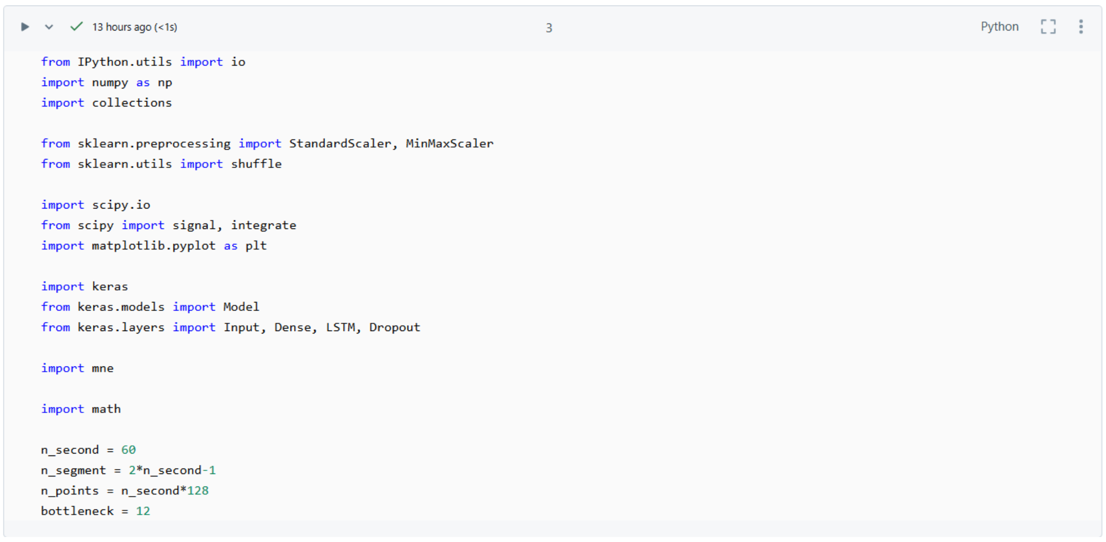


```
%pip install sklearn
%pip install matplotlib
%pip install tensorflow
%pip install numpy>=1.23
%pip install scipy>=1.9
%pip install mne==1.4.0
%pip install tensorflow==2.12.0
```

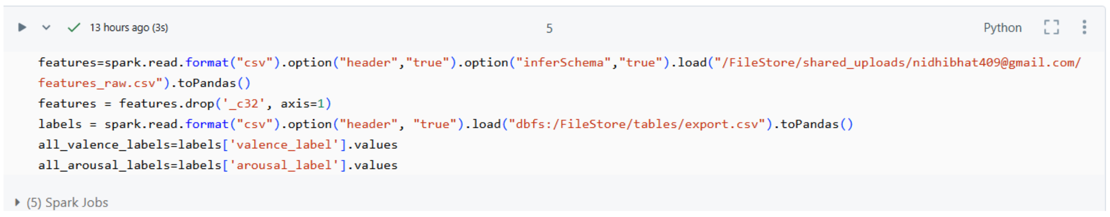


```
import tensorflow as tf
from tensorflow.keras import layers

print("TensorFlow version:", tf.__version__)
print("Keras version:", tf.keras.__version__)

# Simple test model
model = tf.keras.Sequential([
    layers.Input(shape=(10,)),
    layers.Dense(5, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])
model.summary()
```

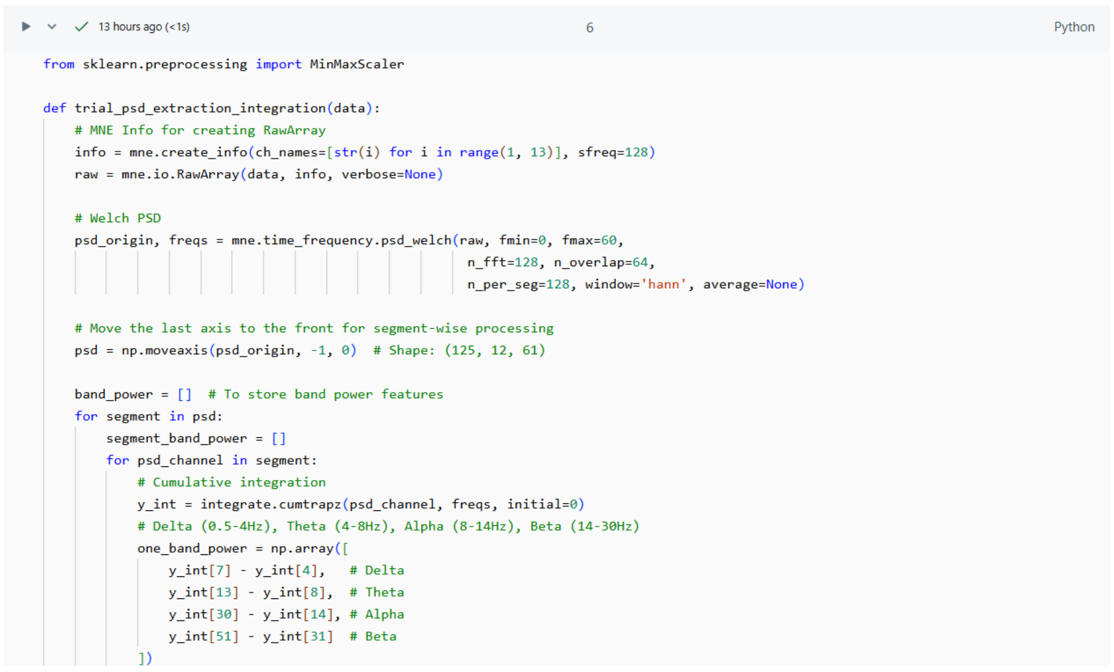
```
TensorFlow version: 2.12.0
Keras version: 2.12.0
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 5)                 55

 dense_1 (Dense)             (None, 1)                 6

=================================================================
Total params: 61
Trainable params: 61
Non-trainable params: 0
```



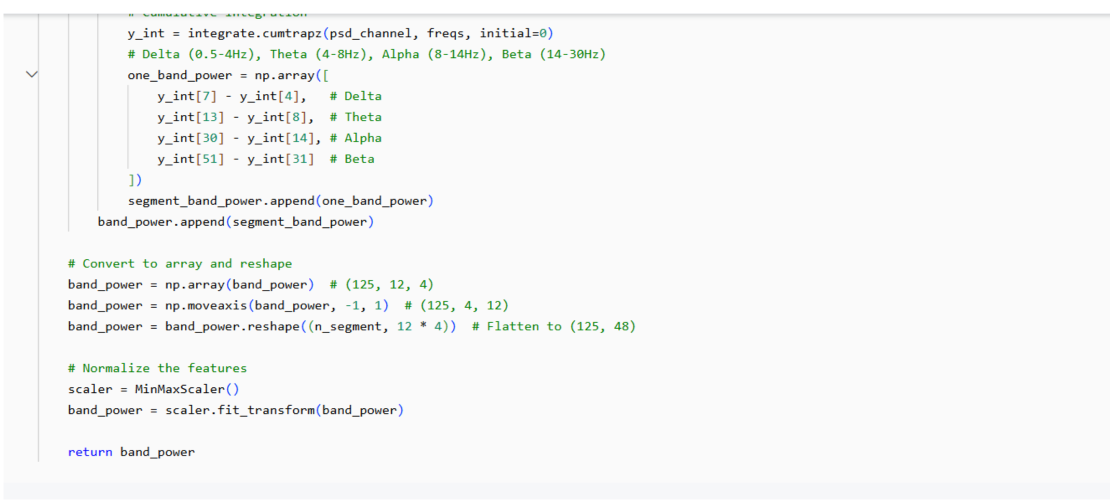
```
from IPython.utils import io
import numpy as np
import collections

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 math

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



```
features=spark.read.format("csv").option("header","true").option("inferSchema","true").load("/FileStore/shared_uploads/nidhibhat409@gmail.com/
features_raw.csv").toPandas()
features = features.drop('_c32', axis=1)
labels = spark.read.format("csv").option("header", "true").load("dbfs:/FileStore/tables/export.csv").toPandas()
all_valence_labels=labels['valence_label'].values
all_arousal_labels=labels['arousal_label'].values
```

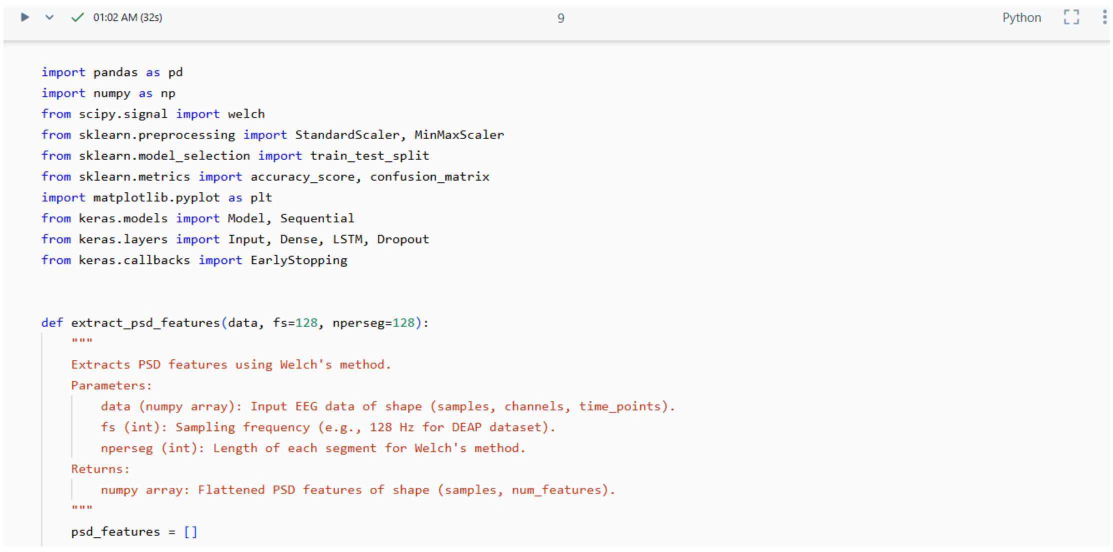

13 hours ago (<1s) 6 Python

```python
from sklearn.preprocessing import MinMaxScaler

def trial_psd_extraction_integration(data):
    # MNE Info for creating RawArray
    info = mne.create_info(ch_names=[str(i) for i in range(1, 13)], sfreq=128)
    raw = mne.io.RawArray(data, info, verbose=None)

    # Welch PSD
    psd_origin, freqs = mne.time_frequency.psd_welch(raw, fmin=0, fmax=60,
                                                     n_fft=128, n_overlap=64,
                                                     n_per_seg=128, window='hann', average=None)

    # Move the last axis to the front for segment-wise processing
    psd = np.moveaxis(psd_origin, -1, 0)  # Shape: (125, 12, 61)

    band_power = []  # To store band power features
    for segment in psd:
        segment_band_power = []
        for psd_channel in segment:
            # Cumulative integration
            y_int = integrate.cumtrapz(psd_channel, freqs, initial=0)
            # Delta (0.5-4Hz), Theta (4-8Hz), Alpha (8-14Hz), Beta (14-30Hz)
            one_band_power = np.array([
                y_int[7] - y_int[4],   # Delta
                y_int[13] - y_int[8],  # Theta
                y_int[30] - y_int[14], # Alpha
                y_int[51] - y_int[31]  # Beta
            ])
```

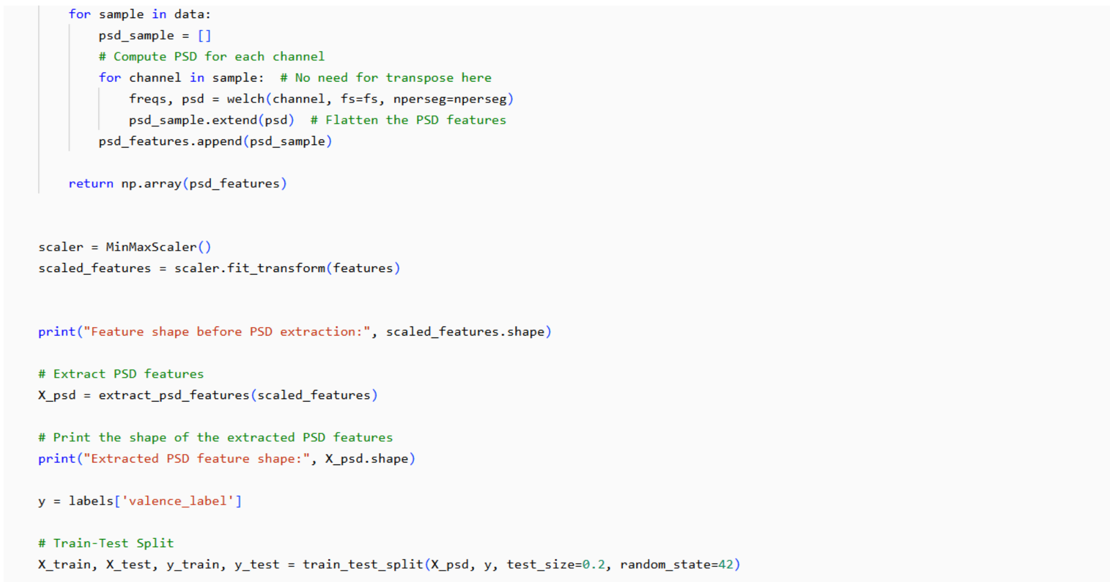


```python
            y_int = integrate.cumtrapz(psd_channel, freqs, initial=0)
            # Delta (0.5-4Hz), Theta (4-8Hz), Alpha (8-14Hz), Beta (14-30Hz)
            one_band_power = np.array([
                y_int[7] - y_int[4],   # Delta
                y_int[13] - y_int[8],  # Theta
                y_int[30] - y_int[14], # Alpha
                y_int[51] - y_int[31]  # Beta
            ])
```

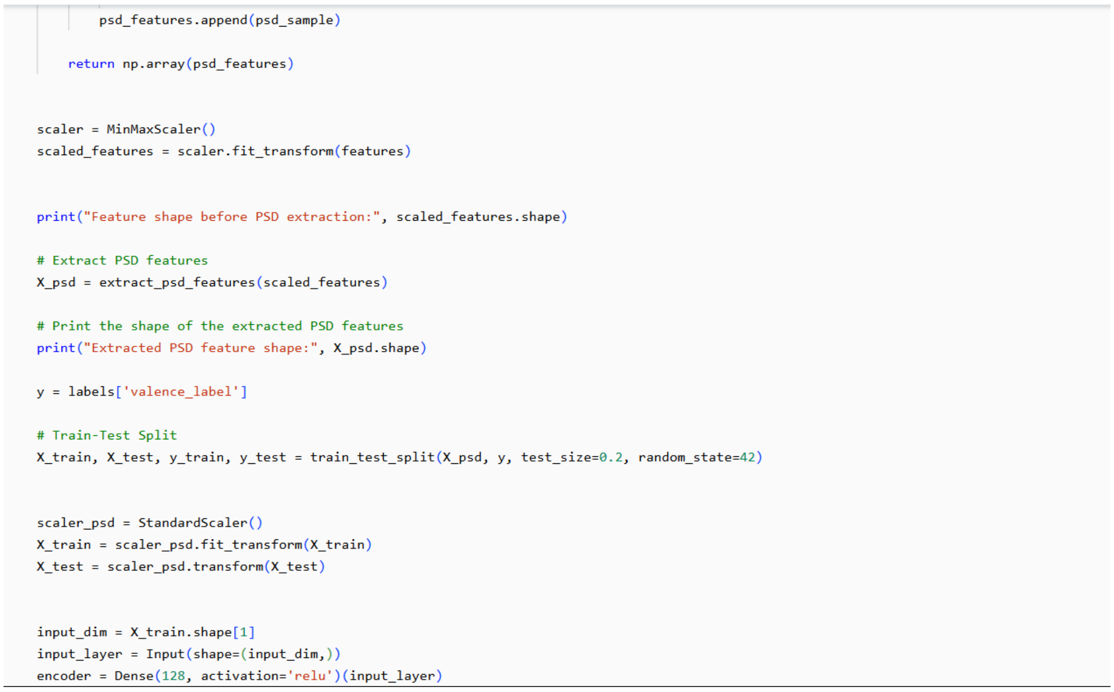
```python
            segment_band_power.append(one_band_power)
        band_power.append(segment_band_power)

    # Convert to array and reshape
    band_power = np.array(band_power)  # (125, 12, 4)
    band_power = np.moveaxis(band_power, -1, 1)  # (125, 4, 12)
    band_power = band_power.reshape((n_segment, 12 * 4))  # Flatten to (125, 48)

    # Normalize the features
    scaler = MinMaxScaler()
    band_power = scaler.fit_transform(band_power)

    return band_power
```


01:02 AM (32s) 9 Python

```python
import pandas as pd
import numpy as np
from scipy.signal import welch
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
from keras.models import Model, Sequential
from keras.layers import Input, Dense, LSTM, Dropout
from keras.callbacks import EarlyStopping


def extract_psd_features(data, fs=128, nperseg=128):
    """
    Extracts PSD features using Welch's method.
    Parameters:
        data (numpy array): Input EEG data of shape (samples, channels, time_points).
        fs (int): Sampling frequency (e.g., 128 Hz for DEAP dataset).
        nperseg (int): Length of each segment for Welch's method.
    Returns:
        numpy array: Flattened PSD features of shape (samples, num_features).
    """
    psd_features = []
```



```python
    for sample in data:
        psd_sample = []
        # Compute PSD for each channel
        for channel in sample:  # No need for transpose here
            freqs, psd = welch(channel, fs=fs, nperseg=nperseg)
            psd_sample.extend(psd)  # Flatten the PSD features
        psd_features.append(psd_sample)

    return np.array(psd_features)


scaler = MinMaxScaler()
scaled_features = scaler.fit_transform(features)


print("Feature shape before PSD extraction:", scaled_features.shape)

# Extract PSD features
X_psd = extract_psd_features(scaled_features)

# Print the shape of the extracted PSD features
print("Extracted PSD feature shape:", X_psd.shape)

y = labels['valence_label']

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_psd, y, test_size=0.2, random_state=42)
```

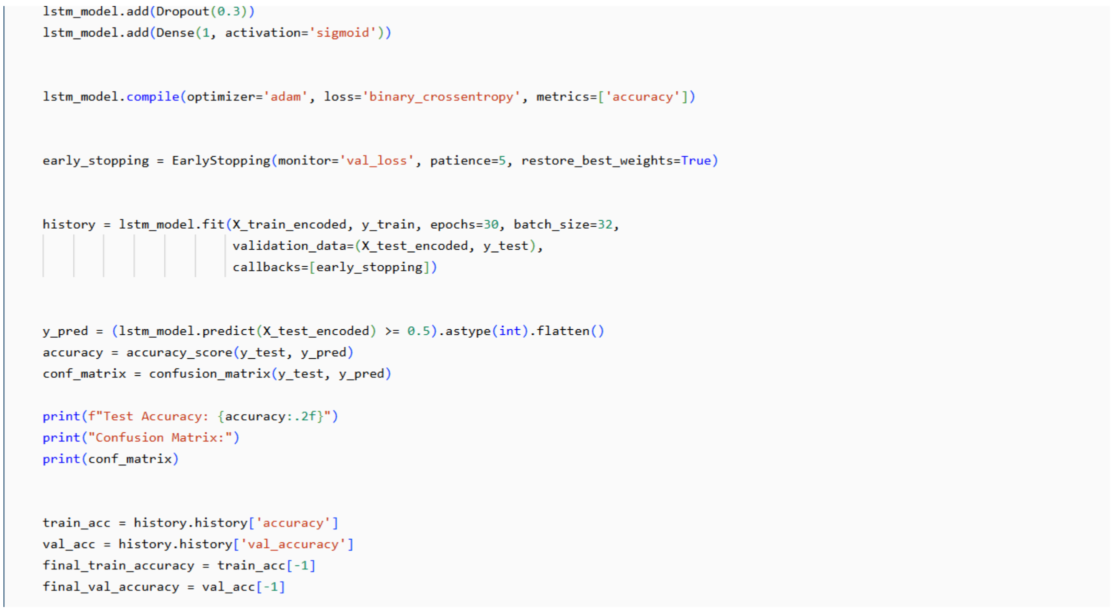


```python
        psd_features.append(psd_sample)

    return np.array(psd_features)


scaler = MinMaxScaler()
scaled_features = scaler.fit_transform(features)


print("Feature shape before PSD extraction:", scaled_features.shape)

# Extract PSD features
X_psd = extract_psd_features(scaled_features)

# Print the shape of the extracted PSD features
print("Extracted PSD feature shape:", X_psd.shape)

y = labels['valence_label']

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_psd, y, test_size=0.2, random_state=42)
```

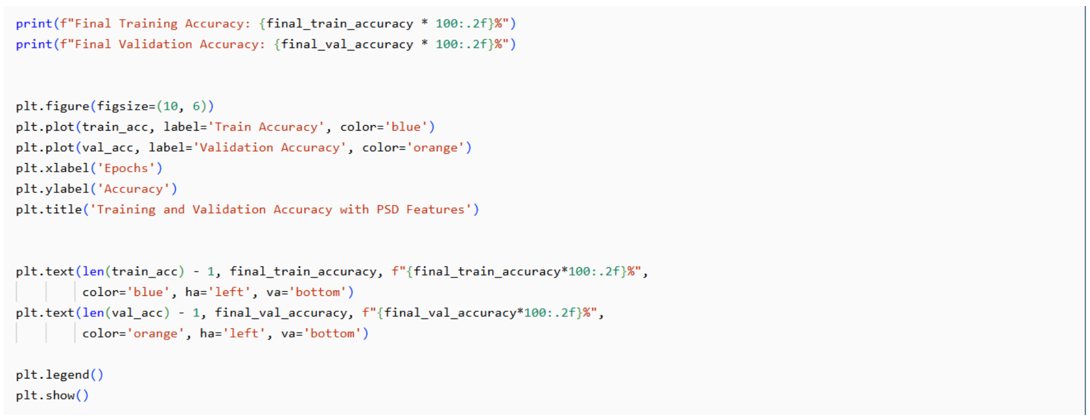
```python
scaler_psd = StandardScaler()
X_train = scaler_psd.fit_transform(X_train)
X_test = scaler_psd.transform(X_test)


input_dim = X_train.shape[1]
input_layer = Input(shape=(input_dim,))
encoder = Dense(128, activation='relu')(input_layer)
```

```
        psd_features.append(psd_sample)

    return np.array(psd_features)


scaler = MinMaxScaler()
scaled_features = scaler.fit_transform(features)


print("Feature shape before PSD extraction:", scaled_features.shape)

# Extract PSD features
X_psd = extract_psd_features(scaled_features)

# Print the shape of the extracted PSD features
print("Extracted PSD feature shape:", X_psd.shape)

y = labels['valence_label']

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_psd, y, test_size=0.2, random_state=42)


scaler_psd = StandardScaler()
X_train = scaler_psd.fit_transform(X_train)
X_test = scaler_psd.transform(X_test)


input_dim = X_train.shape[1]
input_layer = Input(shape=(input_dim,))
encoder = Dense(128, activation='relu')(input_layer)
```

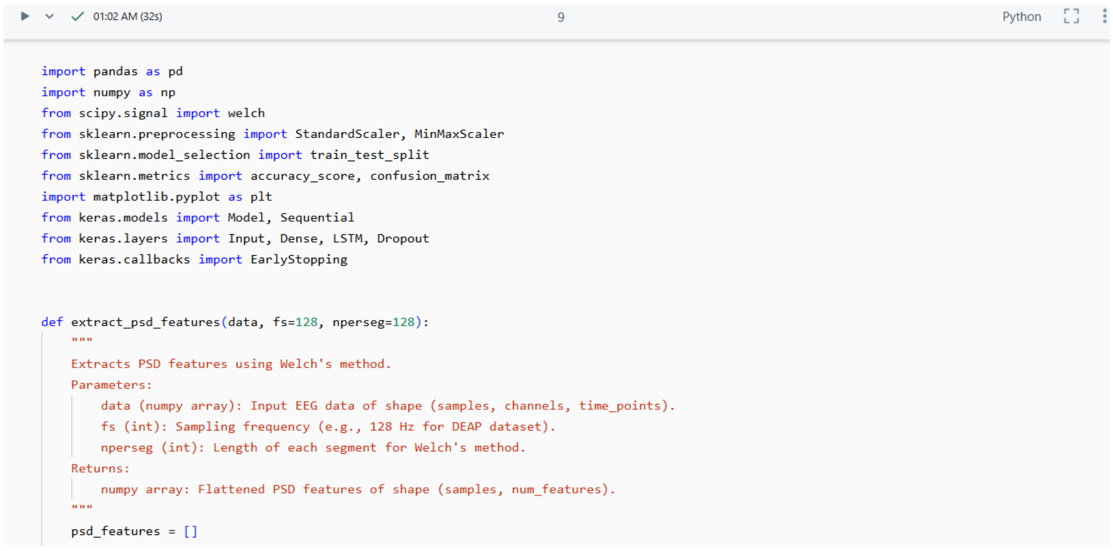
```
decoder_output = Dense(input_dim, activation='sigmoid')(decoder)

autoencoder = Model(input_layer, decoder_output)
autoencoder.compile(optimizer='adam', loss='mse')


autoencoder.fit(X_train, X_train, epochs=10, batch_size=32, shuffle=True, validation_data=(X_test, X_test))


encoder_model = Model(inputs=input_layer, outputs=encoder_output)
X_train_encoded = encoder_model.predict(X_train)
X_test_encoded = encoder_model.predict(X_test)


y_train = y_train.to_numpy().astype(np.float32).reshape(-1, 1)
y_test = y_test.to_numpy().astype(np.float32).reshape(-1, 1)


scaler_encoded = StandardScaler()
X_train_encoded = scaler_encoded.fit_transform(X_train_encoded)
X_test_encoded = scaler_encoded.transform(X_test_encoded)


X_train_encoded = X_train_encoded.reshape(-1, 1, X_train_encoded.shape[1])
X_test_encoded = X_test_encoded.reshape(-1, 1, X_test_encoded.shape[1])


lstm_model = Sequential()
lstm_model.add(LSTM(32, input_shape=(X_train_encoded.shape[1], X_train_encoded.shape[2]), return_sequences=False))
lstm_model.add(Dropout(0.3))
lstm_model.add(Dense(1, activation='sigmoid'))
```



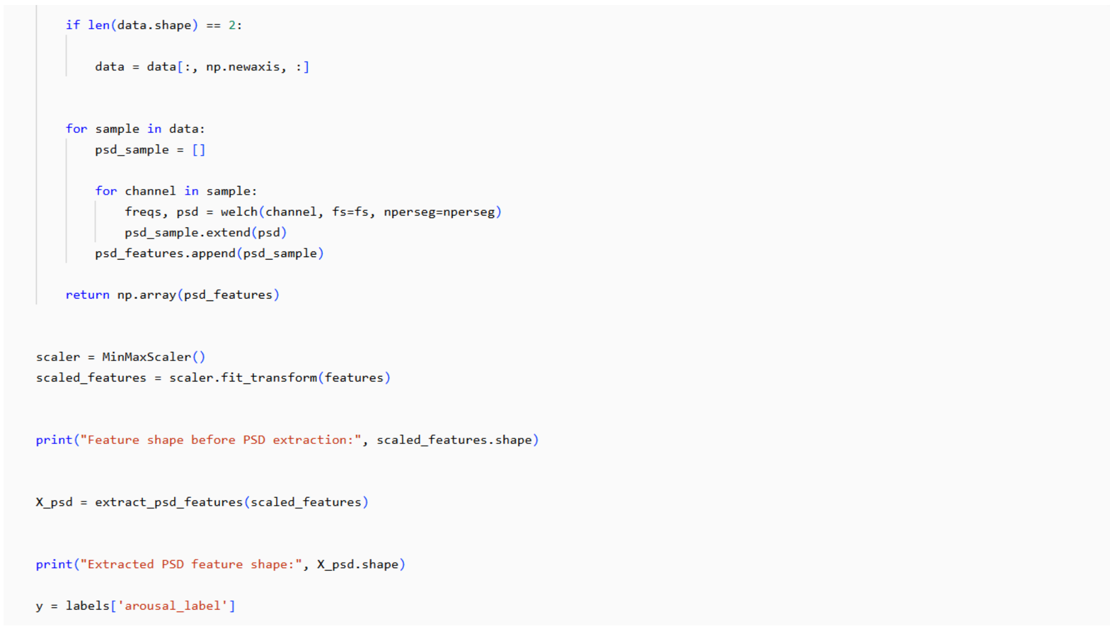
```python
    lstm_model.add(Dropout(0.3))
    lstm_model.add(Dense(1, activation='sigmoid'))


    lstm_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])


    early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)


    history = lstm_model.fit(X_train_encoded, y_train, epochs=30, batch_size=32,
                             validation_data=(X_test_encoded, y_test),
                             callbacks=[early_stopping])


    y_pred = (lstm_model.predict(X_test_encoded) >= 0.5).astype(int).flatten()
    accuracy = accuracy_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)

    print(f"Test Accuracy: {accuracy:.2f}")
    print("Confusion Matrix:")
    print(conf_matrix)


    train_acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    final_train_accuracy = train_acc[-1]
    final_val_accuracy = val_acc[-1]
```



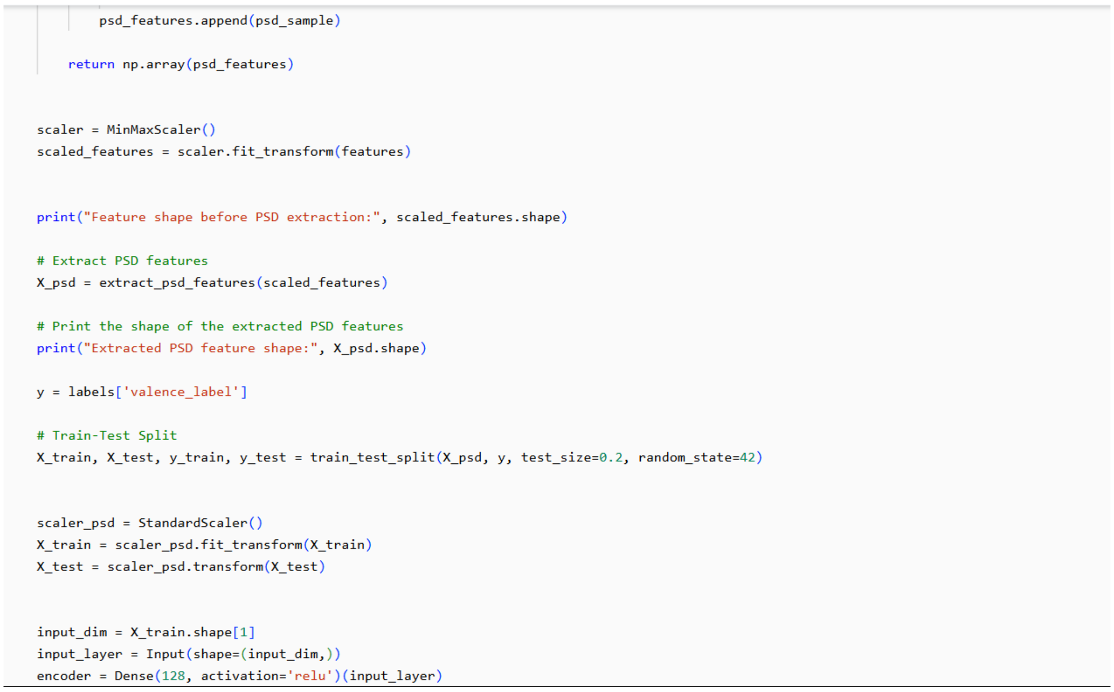
```python
print(f"Final Training Accuracy: {final_train_accuracy * 100:.2f}%")
print(f"Final Validation Accuracy: {final_val_accuracy * 100:.2f}%")


plt.figure(figsize=(10, 6))
plt.plot(train_acc, label='Train Accuracy', color='blue')
plt.plot(val_acc, label='Validation Accuracy', color='orange')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy with PSD Features')


plt.text(len(train_acc) - 1, final_train_accuracy, f"{final_train_accuracy*100:.2f}%",
         color='blue', ha='left', va='bottom')
plt.text(len(val_acc) - 1, final_val_accuracy, f"{final_val_accuracy*100:.2f}%",
         color='orange', ha='left', va='bottom')

plt.legend()
plt.show()
```



```python
    import pandas as pd
    import numpy as np
    from scipy.signal import welch
    from sklearn.preprocessing import StandardScaler, MinMaxScaler
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import accuracy_score, confusion_matrix
    import matplotlib.pyplot as plt
    from keras.models import Model, Sequential
    from keras.layers import Input, Dense, LSTM, Dropout
    from keras.callbacks import EarlyStopping


    def extract_psd_features(data, fs=128, nperseg=128):
        """
        Extracts PSD features using Welch's method.
        Parameters:
            data (numpy array): Input EEG data of shape (samples, channels, time_points).
            fs (int): Sampling frequency (e.g., 128 Hz for DEAP dataset).
            nperseg (int): Length of each segment for Welch's method.
        Returns:
            numpy array: Flattened PSD features of shape (samples, num_features).
        """
        psd_features = []
```



```
    if len(data.shape) == 2:

        data = data[:, np.newaxis, :]


    for sample in data:
        psd_sample = []

        for channel in sample:
            freqs, psd = welch(channel, fs=fs, nperseg=nperseg)
            psd_sample.extend(psd)
        psd_features.append(psd_sample)

    return np.array(psd_features)


scaler = MinMaxScaler()
scaled_features = scaler.fit_transform(features)


print("Feature shape before PSD extraction:", scaled_features.shape)


X_psd = extract_psd_features(scaled_features)


print("Extracted PSD feature shape:", X_psd.shape)

y = labels['arousal_label']
```



```
        psd_features.append(psd_sample)

    return np.array(psd_features)


scaler = MinMaxScaler()
scaled_features = scaler.fit_transform(features)


print("Feature shape before PSD extraction:", scaled_features.shape)

# Extract PSD features
X_psd = extract_psd_features(scaled_features)

# Print the shape of the extracted PSD features
print("Extracted PSD feature shape:", X_psd.shape)

y = labels['valence_label']

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_psd, y, test_size=0.2, random_state=42)


scaler_psd = StandardScaler()
X_train = scaler_psd.fit_transform(X_train)
X_test = scaler_psd.transform(X_test)


input_dim = X_train.shape[1]
input_layer = Input(shape=(input_dim,))
encoder = Dense(128, activation='relu')(input_layer)
```

```
        psd_features.append(psd_sample)

    return np.array(psd_features)


scaler = MinMaxScaler()
scaled_features = scaler.fit_transform(features)


print("Feature shape before PSD extraction:", scaled_features.shape)

# Extract PSD features
X_psd = extract_psd_features(scaled_features)

# Print the shape of the extracted PSD features
print("Extracted PSD feature shape:", X_psd.shape)

y = labels['valence_label']

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_psd, y, test_size=0.2, random_state=42)


scaler_psd = StandardScaler()
X_train = scaler_psd.fit_transform(X_train)
X_test = scaler_psd.transform(X_test)


input_dim = X_train.shape[1]
input_layer = Input(shape=(input_dim,))
encoder = Dense(128, activation='relu')(input_layer)
```

```
decoder_output = Dense(input_dim, activation='sigmoid')(decoder)

autoencoder = Model(input_layer, decoder_output)
autoencoder.compile(optimizer='adam', loss='mse')


autoencoder.fit(X_train, X_train, epochs=10, batch_size=32, shuffle=True, validation_data=(X_test, X_test))


encoder_model = Model(inputs=input_layer, outputs=encoder_output)
X_train_encoded = encoder_model.predict(X_train)
X_test_encoded = encoder_model.predict(X_test)


y_train = y_train.to_numpy().astype(np.float32).reshape(-1, 1)
y_test = y_test.to_numpy().astype(np.float32).reshape(-1, 1)


scaler_encoded = StandardScaler()
X_train_encoded = scaler_encoded.fit_transform(X_train_encoded)
X_test_encoded = scaler_encoded.transform(X_test_encoded)


X_train_encoded = X_train_encoded.reshape(-1, 1, X_train_encoded.shape[1])
X_test_encoded = X_test_encoded.reshape(-1, 1, X_test_encoded.shape[1])


lstm_model = Sequential()
lstm_model.add(LSTM(32, input_shape=(X_train_encoded.shape[1], X_train_encoded.shape[2]), return_sequences=False))
lstm_model.add(Dropout(0.3))
lstm_model.add(Dense(1, activation='sigmoid'))
```

```
    lstm_model.add(Dropout(0.3))
    lstm_model.add(Dense(1, activation='sigmoid'))


    lstm_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])


    early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)


    history = lstm_model.fit(X_train_encoded, y_train, epochs=30, batch_size=32,
                             validation_data=(X_test_encoded, y_test),
                             callbacks=[early_stopping])


    y_pred = (lstm_model.predict(X_test_encoded) >= 0.5).astype(int).flatten()
    accuracy = accuracy_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)

    print(f"Test Accuracy: {accuracy:.2f}")
    print("Confusion Matrix:")
    print(conf_matrix)


    train_acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    final_train_accuracy = train_acc[-1]
    final_val_accuracy = val_acc[-1]
```

```
print(f"Final Training Accuracy: {final_train_accuracy * 100:.2f}%")
print(f"Final Validation Accuracy: {final_val_accuracy * 100:.2f}%")


plt.figure(figsize=(10, 6))
plt.plot(train_acc, label='Train Accuracy', color='blue')
plt.plot(val_acc, label='Validation Accuracy', color='orange')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy with PSD Features')


plt.text(len(train_acc) - 1, final_train_accuracy, f"{final_train_accuracy*100:.2f}%",
         color='blue', ha='left', va='bottom')
plt.text(len(val_acc) - 1, final_val_accuracy, f"{final_val_accuracy*100:.2f}%",
         color='orange', ha='left', va='bottom')

plt.legend()
plt.show()
```

```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.cluster import KMeans
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import keras
from keras.models import Model
from keras.layers import Input, Dense, LSTM, Dropout
from keras.optimizers import Adam
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout

import matplotlib.pyplot as plt
# Step 2: Data Normalization
scaler = MinMaxScaler()
scaled_features = scaler.fit_transform(features.iloc[:, :])  # Exclude labels for scaling
import pandas as pd
# Step 3: Train-Test Split
X = pd.DataFrame(scaled_features)
y = labels['valence_label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 4: Autoencoder Definition
input_dim = X_train.shape[1]
```

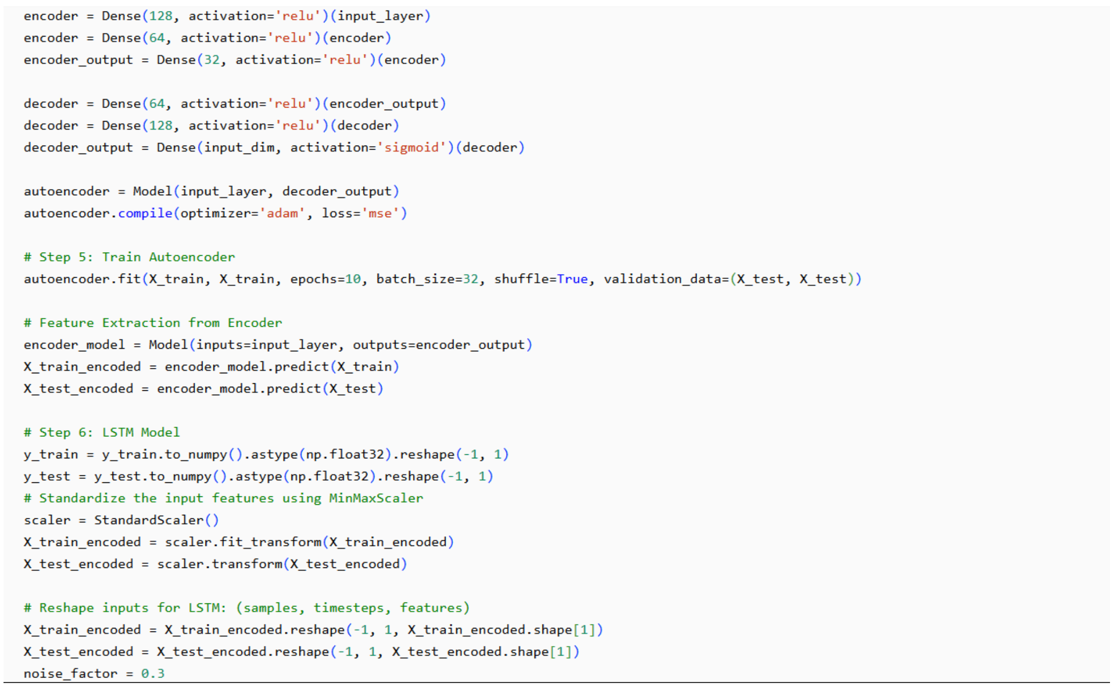
```python
encoder = Dense(128, activation='relu')(input_layer)
encoder = Dense(64, activation='relu')(encoder)
encoder_output = Dense(32, activation='relu')(encoder)

decoder = Dense(64, activation='relu')(encoder_output)
decoder = Dense(128, activation='relu')(decoder)
decoder_output = Dense(input_dim, activation='sigmoid')(decoder)

autoencoder = Model(input_layer, decoder_output)
autoencoder.compile(optimizer='adam', loss='mse')

# Step 5: Train Autoencoder
autoencoder.fit(X_train, X_train, epochs=10, batch_size=32, shuffle=True, validation_data=(X_test, X_test))

# Feature Extraction from Encoder
encoder_model = Model(inputs=input_layer, outputs=encoder_output)
X_train_encoded = encoder_model.predict(X_train)
X_test_encoded = encoder_model.predict(X_test)

# Step 6: LSTM Model
y_train = y_train.to_numpy().astype(np.float32).reshape(-1, 1)
y_test = y_test.to_numpy().astype(np.float32).reshape(-1, 1)
# Standardize the input features using MinMaxScaler
scaler = StandardScaler()
X_train_encoded = scaler.fit_transform(X_train_encoded)
X_test_encoded = scaler.transform(X_test_encoded)

# Reshape inputs for LSTM: (samples, timesteps, features)
X_train_encoded = X_train_encoded.reshape(-1, 1, X_train_encoded.shape[1])
X_test_encoded = X_test_encoded.reshape(-1, 1, X_test_encoded.shape[1])
noise_factor = 0.3
```

```python
X_train_noisy = X_train_encoded + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=X_train_encoded.shape)
X_test_noisy = X_test_encoded + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=X_test_encoded.shape)

# Define LSTM Model
lstm_model = Sequential()
lstm_model.add(LSTM(32, input_shape=(X_train_noisy.shape[1], X_train_noisy.shape[2])))
lstm_model.add(Dropout(0.3))  # Increase dropout rate
lstm_model.add(Dense(1, activation='sigmoid'))


# Compile Model
lstm_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the LSTM Model
from keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Train the model with early stopping
history = lstm_model.fit(X_train_noisy, y_train, epochs=30, batch_size=32,
                         validation_data=(X_test_noisy, y_test),
                         callbacks=[early_stopping])

# Step 8: Evaluation
y_pred = (lstm_model.predict(X_test_noisy) >= 0.5).astype(int).flatten()
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f"Test Accuracy: {accuracy:.2f}")
print("Confusion Matrix:")
print(conf_matrix)
```

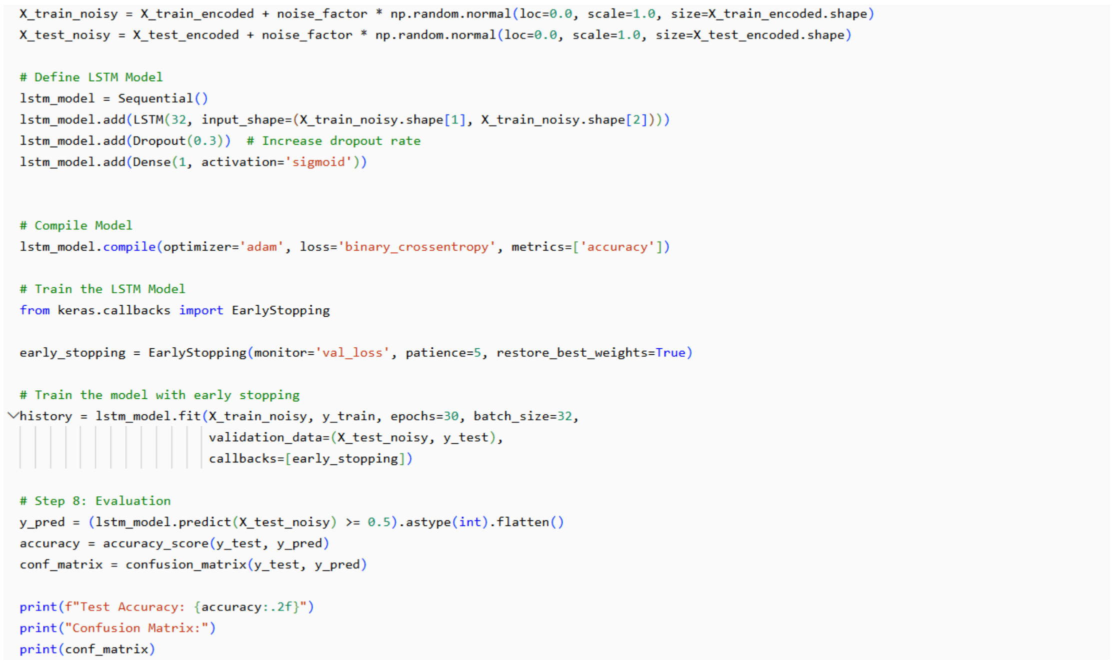
```
train_acc = history.history['accuracy']  # Training accuracy
val_acc = history.history['val_accuracy']
final_train_accuracy = train_acc[-1]  # Last epoch's training accuracy
final_val_accuracy = val_acc[-1]  # Last epoch's validation accuracy

print(f"Final Training Accuracy: {final_train_accuracy * 100:.2f}%")
print(f"Final Validation Accuracy: {final_val_accuracy * 100:.2f}%")

# Optionally, add the final accuracy to the plot
plt.plot(train_acc, label='Train Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')

# Adding text on the plot to show final accuracy
plt.text(len(train_acc)-1, final_train_accuracy, f"{final_train_accuracy*100:.2f}%", color='blue', ha='left', va='bottom')
plt.text(len(val_acc)-1, final_val_accuracy, f"{final_val_accuracy*100:.2f}%", color='orange', ha='left', va='bottom')

plt.legend()
plt.show()
```

```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.cluster import KMeans
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import keras
from keras.models import Model
from keras.layers import Input, Dense, LSTM, Dropout
from keras.optimizers import Adam
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout

import matplotlib.pyplot as plt
# Step 2: Data Normalization
scaler = MinMaxScaler()
scaled_features = scaler.fit_transform(features.iloc[:, :])  # Exclude labels for scaling
import pandas as pd
# Step 3: Train-Test Split
X = pd.DataFrame(scaled_features)
y = labels['valence_label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 4: Autoencoder Definition
input_dim = X_train.shape[1]
```

```
X = pd.DataFrame(scaled_features)
y = labels['arousal_label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 4: Autoencoder Definition
input_dim = X_train.shape[1]

input_layer = Input(shape=(input_dim,))
encoder = Dense(128, activation='relu')(input_layer)
encoder = Dense(64, activation='relu')(encoder)
encoder_output = Dense(32, activation='relu')(encoder)
```



```python
encoder = Dense(128, activation='relu')(input_layer)
encoder = Dense(64, activation='relu')(encoder)
encoder_output = Dense(32, activation='relu')(encoder)

decoder = Dense(64, activation='relu')(encoder_output)
decoder = Dense(128, activation='relu')(decoder)
decoder_output = Dense(input_dim, activation='sigmoid')(decoder)

autoencoder = Model(input_layer, decoder_output)
autoencoder.compile(optimizer='adam', loss='mse')

# Step 5: Train Autoencoder
autoencoder.fit(X_train, X_train, epochs=10, batch_size=32, shuffle=True, validation_data=(X_test, X_test))

# Feature Extraction from Encoder
encoder_model = Model(inputs=input_layer, outputs=encoder_output)
X_train_encoded = encoder_model.predict(X_train)
X_test_encoded = encoder_model.predict(X_test)

# Step 6: LSTM Model
y_train = y_train.to_numpy().astype(np.float32).reshape(-1, 1)
y_test = y_test.to_numpy().astype(np.float32).reshape(-1, 1)
# Standardize the input features using MinMaxScaler
scaler = StandardScaler()
X_train_encoded = scaler.fit_transform(X_train_encoded)
X_test_encoded = scaler.transform(X_test_encoded)

# Reshape inputs for LSTM: (samples, timesteps, features)
X_train_encoded = X_train_encoded.reshape(-1, 1, X_train_encoded.shape[1])
X_test_encoded = X_test_encoded.reshape(-1, 1, X_test_encoded.shape[1])
noise_factor = 0.3
```



```python
X_train_noisy = X_train_encoded + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=X_train_encoded.shape)
X_test_noisy = X_test_encoded + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=X_test_encoded.shape)

# Define LSTM Model
lstm_model = Sequential()
lstm_model.add(LSTM(32, input_shape=(X_train_noisy.shape[1], X_train_noisy.shape[2])))
lstm_model.add(Dropout(0.3))  # Increase dropout rate
lstm_model.add(Dense(1, activation='sigmoid'))


# Compile Model
lstm_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the LSTM Model
from keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Train the model with early stopping
history = lstm_model.fit(X_train_noisy, y_train, epochs=30, batch_size=32,
                         validation_data=(X_test_noisy, y_test),
                         callbacks=[early_stopping])

# Step 8: Evaluation
y_pred = (lstm_model.predict(X_test_noisy) >= 0.5).astype(int).flatten()
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f"Test Accuracy: {accuracy:.2f}")
print("Confusion Matrix:")
print(conf_matrix)
print(conf_matrix)
train_acc = history.history['accuracy']  # Training accuracy
val_acc = history.history['val_accuracy']
final_train_accuracy = train_acc[-1]  # Last epoch's training accuracy
final_val_accuracy = val_acc[-1]  # Last epoch's validation accuracy

print(f"Final Training Accuracy: {final_train_accuracy * 100:.2f}%")
print(f"Final Validation Accuracy: {final_val_accuracy * 100:.2f}%")

# Optionally, add the final accuracy to the plot
plt.plot(train_acc, label='Train Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')

# Adding text on the plot to show final accuracy
plt.text(len(train_acc)-1, final_train_accuracy, f"{final_train_accuracy*100:.2f}%", color='blue', ha='left', va='bottom')
plt.text(len(val_acc)-1, final_val_accuracy, f"{final_val_accuracy*100:.2f}%", color='orange', ha='left', va='bottom')

plt.legend()
plt.show()
```