Contents

List of Figures

List of Tables

Abstract

Electroencephalography (EEG) offers a promising, non-invasive window into neural correlates of emotional states, pivotal for advancing human-computer interaction and affective computing. However, accurate EEG-based emotion recognition faces challenges due to signal complexity, inter-subject variability, and noise. This study introduces ResAttentioNet, a novel deep learning architecture designed for robust emotion classification from EEG signals. The model integrates a Residual Temporal Feature Extractor (RTFE) to capture hierarchical temporal patterns and a Multi-Head Temporal Attention Pooling (MTAP) mechanism to focus on salient EEG segments. Differential Entropy (DE) features, extracted from five standard frequency bands (Delta, Theta, Alpha, Beta, Gamma), serve as input to the model. We evaluated ResAttentioNet's performance on the SEED, SEED-IV, SEED-V, and SEED-VII datasets, which encompass a range of discrete emotional states. The proposed model demonstrated strong classification capabilities across these datasets, achieving accuracies of 100% on SEED, 76% on SEED-IV, 80% on SEED-V, and 62% on SEED-VII. These results, particularly the notable performance on the SEED dataset, highlight the potential of the ResAttentioNet architecture in effectively learning discriminative representations from EEG data. This research contributes to the development of more accurate and reliable EEG-based emotion recognition systems, paving the way for enhanced applications in affective computing.

1 Introduction

Emotions are fundamental to human cognition, significantly influencing decision-making, perception, and social interactions. Recognizing and interpreting emotional states is crucial for advancing human-computer interaction, mental health diagnostics, and adaptive intelligent systems. Among the various modalities for emotion recognition, electroencephalography (EEG) has emerged as a powerful tool due to its direct measurement of brain activity, offering high temporal resolution and non-invasive acquisition.

EEG captures the brain's electrical activity, reflecting dynamic neural processes associated with emotional experiences. Compared to other physiological signals, EEG offers a unique advantage in detecting subtle changes in neural states, making it a reliable source for emotion recognition research. The advent of portable and cost-effective EEG devices has further broadened the scope of real-world applications.

EEG-based emotion recognition has been explored across several domains, including healthcare, education, entertainment, and affective computing. In healthcare, it can assist in monitoring emotional well-being and detecting affective disorders. In educational settings, EEG can help evaluate student engagement and personalize learning experiences. In entertainment and marketing, understanding user emotions enables improved content delivery and user satisfaction.

Despite its potential, EEG-based emotion recognition faces several challenges. EEG signals are highly variable due to individual differences, non-stationary behavior, and environmental factors. Moreover, signal artifacts—such as those caused by muscle movements or eye blinks—can degrade signal quality, necessitating robust preprocessing. The lack of standardized data acquisition protocols and emotion elicitation methods also hinders model generalizability and the reproducibility of research.

Recent advancements in machine learning, especially deep learning, have shown promise in overcoming some of these limitations. Deep learning models can automatically learn hierarchical features from raw EEG data,

potentially improving the accuracy and robustness of recognition systems. However, such models require large and diverse datasets, highlighting the importance of standardized benchmarks and collaborative data sharing.

This study explores the current state of EEG-based emotion recognition, with a focus on deep learning methodologies. By analyzing recent literature and identifying key challenges, we aim to provide insights into future directions for the development of reliable EEG-based affective computing systems.

Literature Review

The use of electroencephalogram (EEG) signals for automated emotion recognition has gained significant interest, offering a quantitative and objective approach to understanding human affective states. EEG signals, which reflect brain electrical activity, provide a rich, non-invasive data source for classifying emotions such as happiness, sadness, anger, and fear. Research in this area has evolved in parallel with advancements in machine learning—from traditional techniques reliant on handcrafted features to end-to-end deep learning frameworks capable of automatic representation learning [1, 2, 3].

Traditional machine learning approaches for EEG-based emotion recognition depend on the extraction of discriminative features. Commonly used features include statistical metrics (mean, variance, skewness, kurtosis), power spectral densities across standard frequency bands (delta, theta, alpha, beta, gamma), Hjorth parameters, entropy measures (e.g., Shannon entropy, sample entropy), fractal dimensions, and inter-channel connectivity features like coherence and phase synchronization [2, 1]. These features are typically fed into classifiers such as Support Vector Machines (SVM), k-Nearest Neighbors (KNN), Random Forests, Naive Bayes, and Linear Discriminant Analysis (LDA). For instance, Zheng and Lu utilized differential entropy features combined with SVM on the SEED dataset to achieve high accuracy [1], while Atkinson and Campos demonstrated the value of feature selection with classifiers like SVM and KNN to enhance performance [2]. However, these methods often depend heavily on expert knowledge for feature extraction and may not generalize well across subjects and recording conditions. Benchmark datasets like SEED and SEED-IV have become instrumental in comparing such approaches [4].

With the emergence of deep learning, EEG-based emotion recognition has seen a paradigm shift. Deep learning models can automatically learn complex, hierarchical representations from EEG data, reducing the need for handcrafted features. Convolutional Neural Networks (CNNs) have been particularly effective in capturing spatial patterns from multi-channel EEG

recordings. For example, Li et al. employed a hierarchical CNN to extract discriminative features from raw EEG for emotion classification [5]. Recurrent Neural Networks (RNNs), including LSTMs and GRUs, are well-suited for modeling temporal dependencies in EEG data, and several studies have demonstrated their superior performance over traditional methods [6].

Hybrid models that integrate CNNs and RNNs (e.g., CNN-LSTM) have proven especially effective, where CNNs extract spatial features and RNNs model temporal sequences [7]. Attention mechanisms have also been incorporated into deep models to selectively focus on the most informative EEG channels and time segments, further improving performance and interpretability [3, 8]. For example, Liu et al. developed a multi-level feature-guided capsule network using the SEED dataset, achieving high accuracy [9].

Transforming EEG data into time-frequency representations (e.g., spectrograms via STFT or wavelet transforms) has enabled the use of CNNs for image-like analysis. This approach capitalizes on CNNs' strong performance in computer vision and has yielded encouraging results [10]. Studies such as those by Yin et al. have also explored domain adaptation and transfer learning to address the variability of EEG signals across subjects and sessions, a critical challenge in this field [8, 11].

There is increasing emphasis on developing robust and generalizable models that perform reliably in real-world applications. Unsupervised and semi-supervised learning approaches are gaining traction to reduce dependence on large labeled datasets [12, 13]. Interpretability in deep learning models is also a growing focus, aiming to provide insights into learned representations and support applications in clinical settings [14].

2 Methodology

All experiments are conducted on the SEED family of datasets, each collected using an EEG setup with 62 electrodes. From this data, Differential Entropy (DE) features are extracted individually for five frequency bands: delta (1–4 Hz), theta (4–8 Hz), alpha (8–14 Hz), beta (14–31 Hz), and gamma (31–50 Hz).

2.1 Data Preprocessing and Feature Extraction

To mitigate the effects of noise in the EEG recordings, we begin by visually inspecting the signals and interpolating bad channels using the MNE-Python toolbox. A bandpass filter with cutoff frequencies of 0.1 Hz and 70 Hz is applied to remove low-frequency drift and high-frequency artifacts. Additionally, a notch filter at 50 Hz is used to eliminate powerline interference. To reduce computational complexity, the raw EEG signals are downsampled from 1000 Hz to 200 Hz.After feature extraction, each trial is temporally aligned to a fixed length(60) by truncating or zero-padding as necessary, ensuring that all input samples have a consistent shape suitable for batch processing in deep learning models.

Following preprocessing, we extract differential entropy (DE) features from the EEG data. DE is computed over five commonly used frequency bands:

(1) Delta (1–4 Hz),

(2) Theta (4–8 Hz),

(3) Alpha (8–14 Hz),

(4) Beta (14–31 Hz), and

(5) Gamma (31–50 Hz).

2.1.1 Differential Entropy (DE) as a Feature for EEG Emotion Recognition

Differential Entropy (DE) has emerged as a robust and discriminative feature for EEG-based emotion recognition [2]. It extends the concept of Shannon entropy to continuous random variables, offering a measure of

the average uncertainty or information content of a continuous distribution. Given a continuous random variable X with a probability density function $P(x)$, the DE is defined as:

$$\mathrm{DE} = -\int_{-\infty}^{\infty} P(x) \ln(P(x))\, dx \tag{1}$$

In the context of affective EEG analysis, DE is particularly effective in capturing the dynamic changes in brain activity associated with emotional states. Due to the non-stationary nature of EEG signals, features like DE that can quantify localized signal variability are highly valuable.

A common assumption in EEG signal processing is that short segments of EEG data can be approximated as Gaussian-distributed. Under this assumption, the DE of a signal segment $x \sim \mathcal{N}(\mu, \sigma^2)$ simplifies to a closed-form expression:

$$\mathrm{DE} = \frac{1}{2} \ln(2\pi e \sigma^2) \tag{2}$$

This simplification makes DE computationally efficient and scalable to large EEG datasets. The computed DE features across all channels and frequency bands form a high-dimensional feature matrix that is subsequently used for emotion classification.

Frequency Band	**Range (Hz)**	**Associated Brain States**
Delta	1–4	Deep sleep, unconscious state
Theta	4–8	Drowsiness, relaxation, meditation
Alpha	8–14	Calm, relaxed but alert mental state
Beta	14–31	Active thinking, focus, problem solving
Gamma	31–50	Cognitive processing, memory, perception

Table 1: Standard EEG Frequency Bands and Their Functional Associations

The widespread use of DE in EEG-based emotion recognition is largely attributed to its sensitivity to the information-rich variations across different frequency bands, which are modulated during emotional experiences [1]. By combining robust preprocessing with DE feature extraction across cognitively relevant frequency bands, we construct a meaningful representation of brain activity for downstream emotion classification tasks.

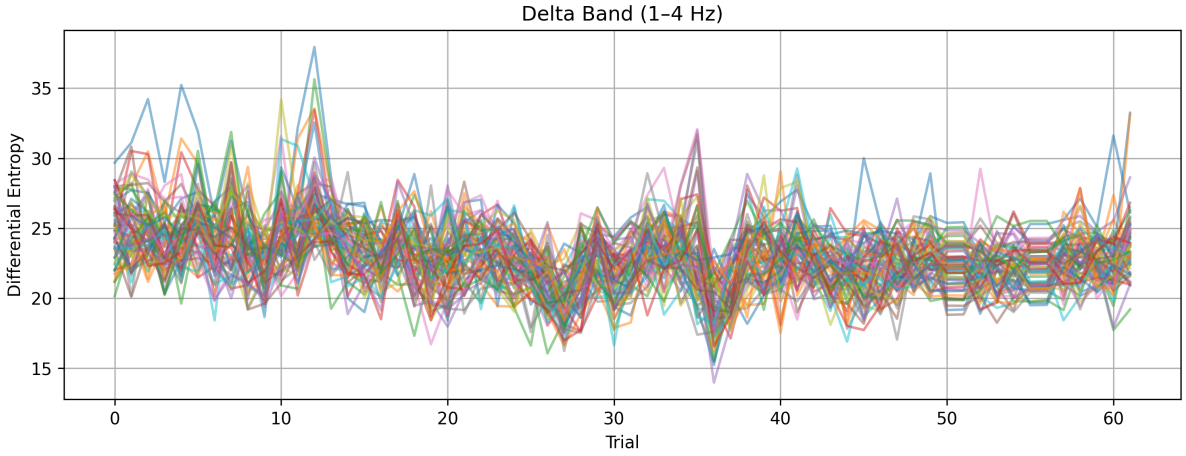


Figure 1: Distribution of Differential Entropy values for the Delta frequency band (1–4 Hz)

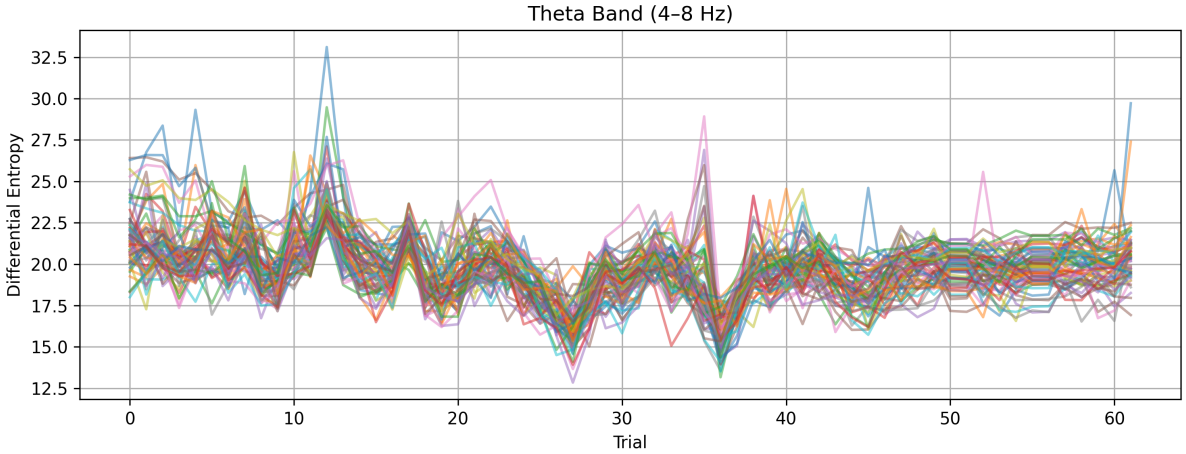


Figure 2: Distribution of Differential Entropy values for the Theta frequency band (4-8 Hz)

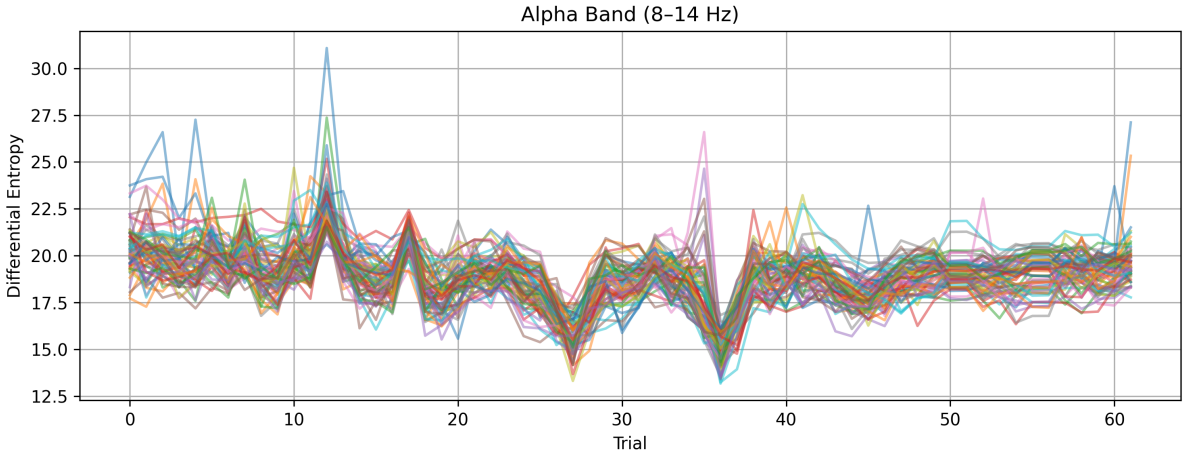


Figure 3: Distribution of Differential Entropy values for the Alpha frequency band (8-14 Hz)

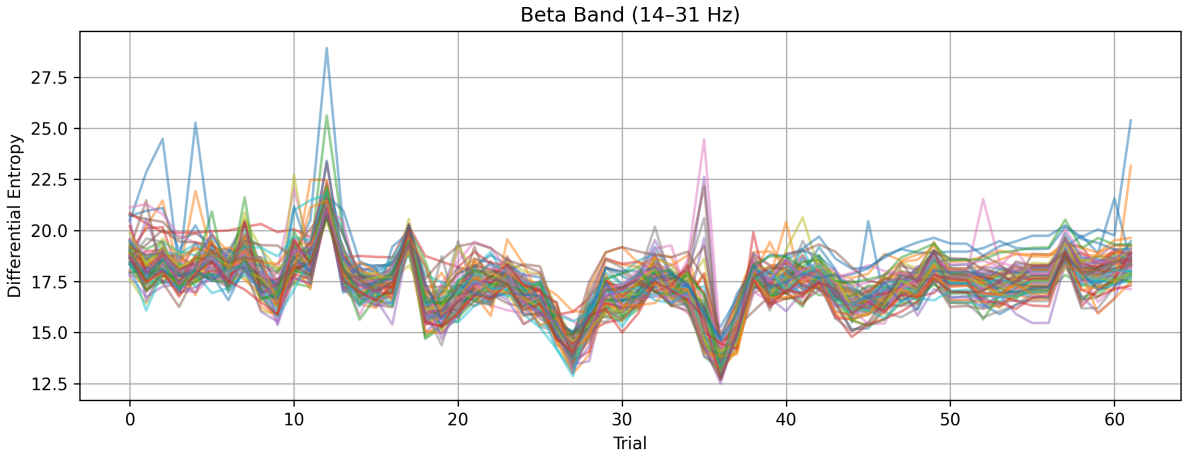


Figure 4: Distribution of Differential Entropy values for the Beta frequency band (14-31 Hz)

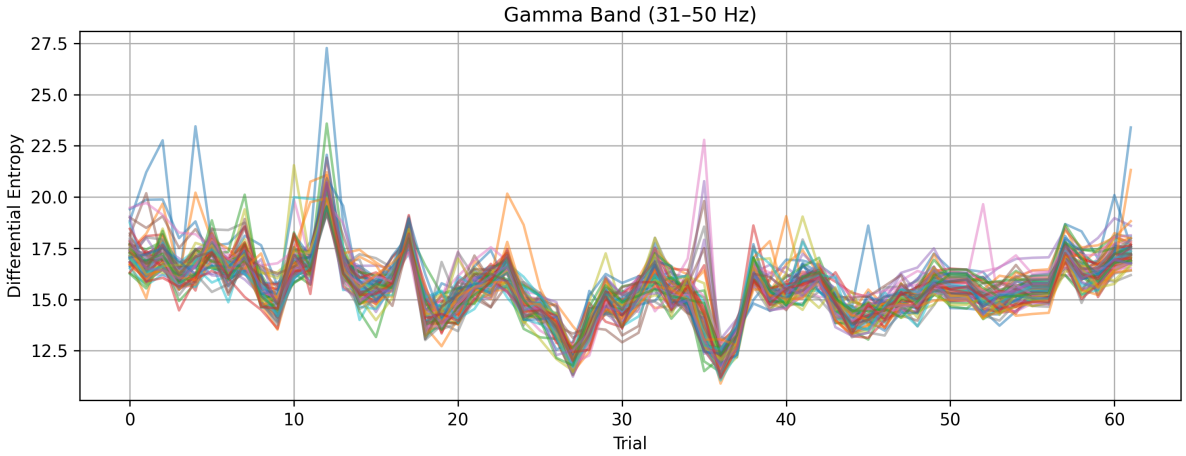


Figure 5: Distribution of Differential Entropy values for the Gamma frequency band (31-50 Hz)

We chose differential entropy (DE) features for EEG-based emotion recognition due to their robustness in capturing the dynamic, non-stationary changes in brain activity [15]. DE extends Shannon entropy to continuous variables, offering a computationally efficient measure of signal uncertainty that simplifies to a closed-form expression under Gaussian assumptions, making it scalable for large datasets [16]. Unlike traditional features such as power spectral density (PSD), Hjorth parameters, or connectivity metrics like coherence, which often require expert knowledge and may not generalize well across subjects, DE is highly sensitive to information-rich variations across frequency bands modulated by emotional states. This sensitivity, combined with its ability to quantify localized signal variability, makes DE a superior choice over statistical metrics, entropy measures, or fractal dimensions, providing a strong foundation for accurate emotion classification when paired with classifiers [17].

2.2 Model Architecture

The input to our model is a preprocessed EEG tensor $\mathbf{X} \in \mathbb{R}^{B\times 60\times 310}$, where B is the batch size, 60 represents temporal steps, and 310 features correspond to signals recorded from 62 electrodes across 5 distinct frequency bands.

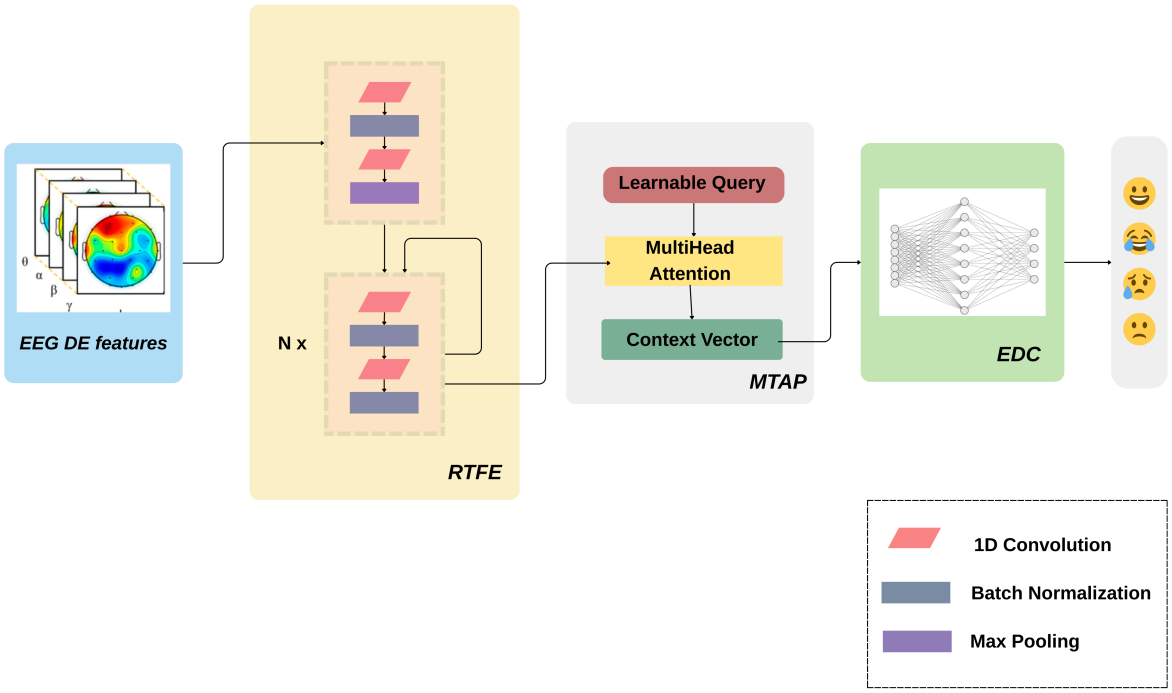


Figure 6: Architecture of ResAttentioNet

2.2.1 Residual Temporal Feature Extractor (RTFE)

The feature extraction stage is built upon a modified ResNet18 architecture adapted for one-dimensional signal processing. The RTFE begins with an initial convolutional layer, followed by batch normalization and ReLU activation. A max-pooling layer then reduces the sequence dimension while preserving temporal structure.

The encoder comprises four residual layers. Each residual block includes two 1D convolutional layers with batch normalization and ReLU activation. The skip connections in each block enable the network to learn residual functions, improving convergence and mitigating the vanishing gradient issue.

2.2.2 Multi-Head Temporal Attention Pooling (MTAP)

After feature extraction, we apply a multi-head temporal attention pooling (MTAP) mechanism to focus on the most informative parts of the EEG

sequence. This module employs a learnable query vector $\mathbf{q} \in \mathbb{R}^D$ that attends to the extracted features $\mathbf{Z} \in \mathbb{R}^{B\times S\times D}$, where S is the reduced sequence length and D is the feature dimension. For each attention head h, the attention weights are computed as:

$$\alpha_{h,i} = \frac{\exp(\mathbf{q}^\top K_{h,i}/\sqrt{D_h})}{\sum_{j=1}^{S}\exp(\mathbf{q}^\top K_{h,j}/\sqrt{D_h})}, \tag{3}$$

$$\mathbf{a}_h = \sum_{i=1}^{S}\alpha_{h,i}V_{h,i}, \tag{4}$$

where K_h and V_h are linear projections of the encoded sequence into key and value vectors, respectively. The outputs of all heads are concatenated and linearly projected to form the pooled representation $\mathbf{z} \in \mathbb{R}^D$.

2.2.3 Emotion Discriminative Classifier (EDC)

The classification module translates the pooled representation into emotion labels. It begins with a dropout layer, followed by a fully connected layer that projects the features to a 128-dimensional hidden space. After another dropout, the final linear layer maps the hidden representation to the target emotion class space. A softmax activation function is applied during training to obtain class probabilities.

2.2.4 *ResAttentioNet*

ResAttentioNet integrates the strengths of residual convolutional encoders and attention-based aggregation to effectively model both short- and long-term dependencies in EEG time series. By leveraging the Residual Temporal Feature Extractor (RTFE), the model captures hierarchical temporal patterns in the signal. The Multi-Head Temporal Attention Pooling (MTAP) component enhances this representation by identifying emotionally salient time steps. Finally, the Emotion Discriminative Classifier (EDC) ensures robust mapping to discrete emotional states.

This design is well-suited for EEG-based emotion recognition tasks, addressing challenges such as low signal-to-noise ratio, inter-subject variability, and the necessity of temporal abstraction. The architecture can be

easily adapted to various class granularities by adjusting the output layer, making it a flexible solution for affective EEG analysis.

The motivation behind designing the ResAttentioNet architecture is to effectively address core challenges in EEG-based emotion recognition, including signal noise, inter-subject variability, and complex temporal dynamics. Its Residual Temporal Feature Extractor (RTFE) employs 1D convolutions and residual connections to capture multi-scale temporal features while ensuring deep network trainability. The Multi-Head Temporal Attention Pooling (MTAP) selectively emphasizes emotionally salient time segments, enhancing feature discriminability. Finally, the Emotion Discriminative Classifier (EDC) maps these features to emotion classes using dropout to improve generalization. Together, these components enable effective modeling of EEG's spatial and temporal characteristics, making ResAttentioNet a robust and adaptable solution for emotion classification from brain signals.

3 Experimental Setup

3.1 Dataset

This project utilizes the SEED (SJTU Emotion EEG Dataset) family of datasets from Shanghai Jiao Tong University, which are extensively employed in emotion recognition research. Specifically, the SEED [1], SEED-IV [18], SEED-V [19], and SEED-VII [20] datasets are considered.

The SEED dataset comprises Electroencephalography (EEG) data and corresponding emotional labels (sad, happy, and neutral) from 15 subjects (seven males and eight females). Each participant underwent three experimental sessions. Each session consisted of 15 trials, culminating in a total of 45 trials per subject. The structure of each trial included a 5-second cue, followed by a 4-minute movie clip intended to elicit the target emotion, a 45-second self-assessment period, and finally, a 15-second rest period.

The SEED-IV dataset involved 15 subjects, each participating in three sessions. Unlike the original SEED dataset, each session in SEED-IV con-

tained 24 trials. Each trial commenced with a 5-second cue, followed by a movie clip lasting between 2 to 4 minutes. After the stimulus, participants engaged in a 45-second self-assessment, reporting one of four emotions: happiness, sadness, fear, or neutral.

Expanding on this, the SEED-V dataset was collected from 16 subjects. Similar to the previous versions, each subject took part in three sessions, with each session encompassing 15 trials, resulting in 45 trials per participant. A trial in SEED-V began with a 15-second cue, followed by a movie clip of 2 to 4 minutes in duration. The self-assessment period was either 15 or 30 seconds. This dataset includes EEG data and corresponding labels for five distinct emotions: disgust, fear, sadness, happiness, and neutral.

The SEED-VII dataset comprises EEG recordings from 20 subjects. Each subject participated in four sessions, with each session containing 20 trials, resulting in a total of 80 trials per participant. The structure of each trial included a 3-second cue at the beginning, followed by a 3-minute emotion-eliciting movie clip, another 3-second cue, and finally a 10-second feedback period. The dataset focuses on the classification of seven discrete emotional states: Neutral, Happy, Sad, Anger, Fear, Disgust, and Surprise.

3.2 Experimental Settings

To evaluate the performance of our proposed architecture, we compare it against several state-of-the-art models, including both single-modal and multi-modal approaches, trained on SEED, SEED IV, SEED V and SEED VII datasets. All experiments were implemented using the PyTorch framework. For each experimental run, the dataset was systematically partitioned into training, validation, and testing subsets. Specifically, 70% of the subjects were allocated to the training set, 15% to the validation set, and the remaining 15% to the testing set. This subject-wise split ensures independence across sets, preventing data leakage and promoting fair evaluation of generalization performance.

4 Results And Discussion

4.1 Performance Parameters

In this study, several key metrics were employed to evaluate the effectiveness of the classification model, which utilized optimally selected EEG features as input. These metrics include Accuracy, Precision, Recall (Sensitivity), and F1-score. Each metric offers unique insight into different aspects of model performance.

Accuracy

Accuracy refers to the overall proportion of correctly classified EEG segments across all emotion classes. It is calculated as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

Alternatively, in a multi-class context using true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) across all classes:

$$\text{Accuracy} = \frac{\sum(\text{TP}_i + \text{TN}_i)}{\sum(\text{TP}_i + \text{TN}_i + \text{FP}_i + \text{FN}_i)}$$

While accuracy is useful for general performance overview, it may not be sufficient in the presence of class imbalance.

Precision (per class)

Precision measures the model's ability to correctly identify only relevant instances for a given class. It is defined as:

$$\text{Precision}_{\text{class}} = \frac{\text{TP}_{\text{class}}}{\text{TP}_{\text{class}} + \text{FP}_{\text{class}}}$$

A high precision indicates a low false positive rate for that emotion class.

Recall (Sensitivity, per class)

Recall, also known as sensitivity or the true positive rate, quantifies the model's ability to identify all actual instances of a specific emotion class:

$$\text{Recall}_{\text{class}} = \frac{\text{TP}_{\text{class}}}{\text{TP}_{\text{class}} + \text{FN}_{\text{class}}}$$

High recall reflects a low false negative rate.

F1-score (per class)

The F1-score is the harmonic mean of precision and recall, providing a balanced metric especially useful when dealing with class imbalance:

$$\text{F1-score}_{\text{class}} = 2 \times \frac{\text{Precision}_{\text{class}} \times \text{Recall}_{\text{class}}}{\text{Precision}_{\text{class}} + \text{Recall}_{\text{class}}}$$

The F1-score ranges from 0 to 1, with 1 indicating perfect precision and recall.

4.2 Insights from Experimental Results

Dataset	Class	Precision	Recall	F1-score
SEED	Negative	1.00	1.00	1.00
	Neutral	1.00	1.00	1.00
	Positive	1.00	1.00	1.00
	Accuracy		**1.00**	
SEED-IV	Disgust	0.75	0.72	0.74
	Sad	0.76	0.72	0.74
	Neutral	0.77	0.74	0.75
	Happy	0.77	0.87	0.82
	Accuracy		**0.76**	
SEED-V	Disgust	1.00	0.89	0.94
	Fear	0.89	0.63	0.74
	Sad	0.62	0.85	0.72
	Neutral	0.81	0.81	0.81
	Happy	0.79	0.81	0.80
	Accuracy		**0.80**	
SEED-VII	Neutral	0.59	0.79	0.68
	Happy	0.72	0.72	0.72
	Sad	0.61	0.31	0.41
	Anger	0.47	0.69	0.56
	Fear	0.52	0.36	0.43
	Disgust	0.90	0.78	0.84
	Surprise	0.62	0.78	0.69
	Accuracy		**0.62**	

We present the key outcomes of the study evaluating the ResAttentioNet architecture for EEG-based emotion recognition across multiple datasets. The efficacy of the proposed ResAttentioNet model was rigorously evaluated against several baseline and state-of-the-art approaches using the SEED, SEED-IV, SEED-V, and SEED-VII datasets. Performance was primarily assessed using accuracy, F1-score, precision, and recall.

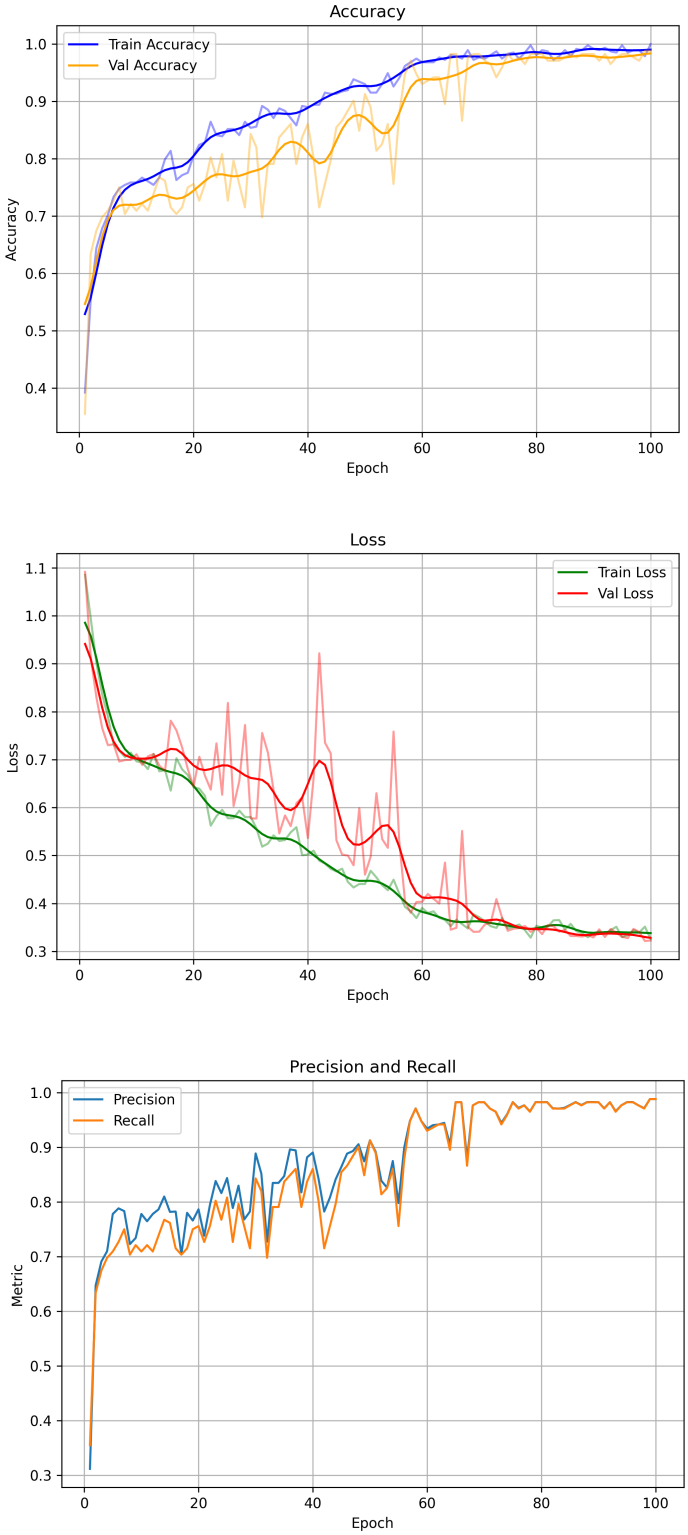


Figure 7: Learning curves for training and validation on the SEED dataset

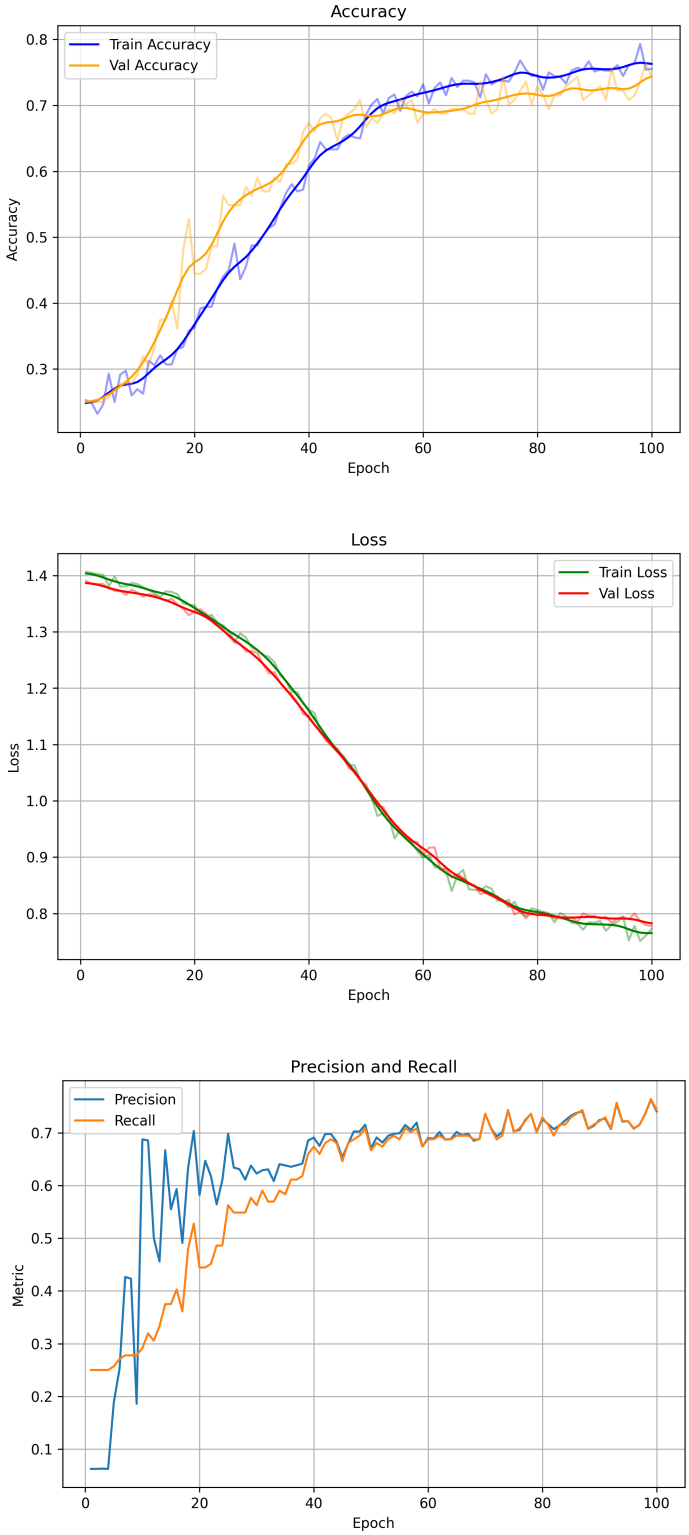


Figure 8: Learning curves for training and validation on the SEED-IV dataset

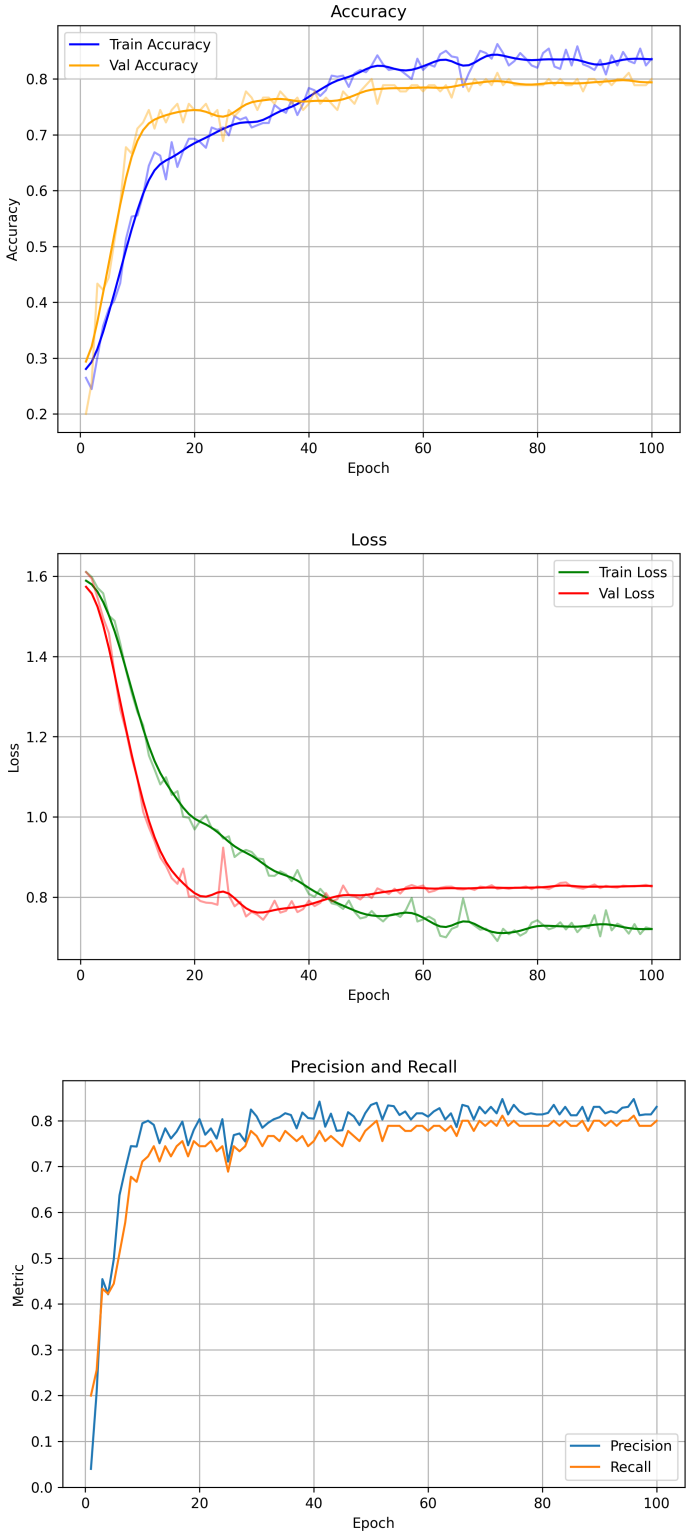


Figure 9: Learning curves for training and validation on the SEED-V dataset

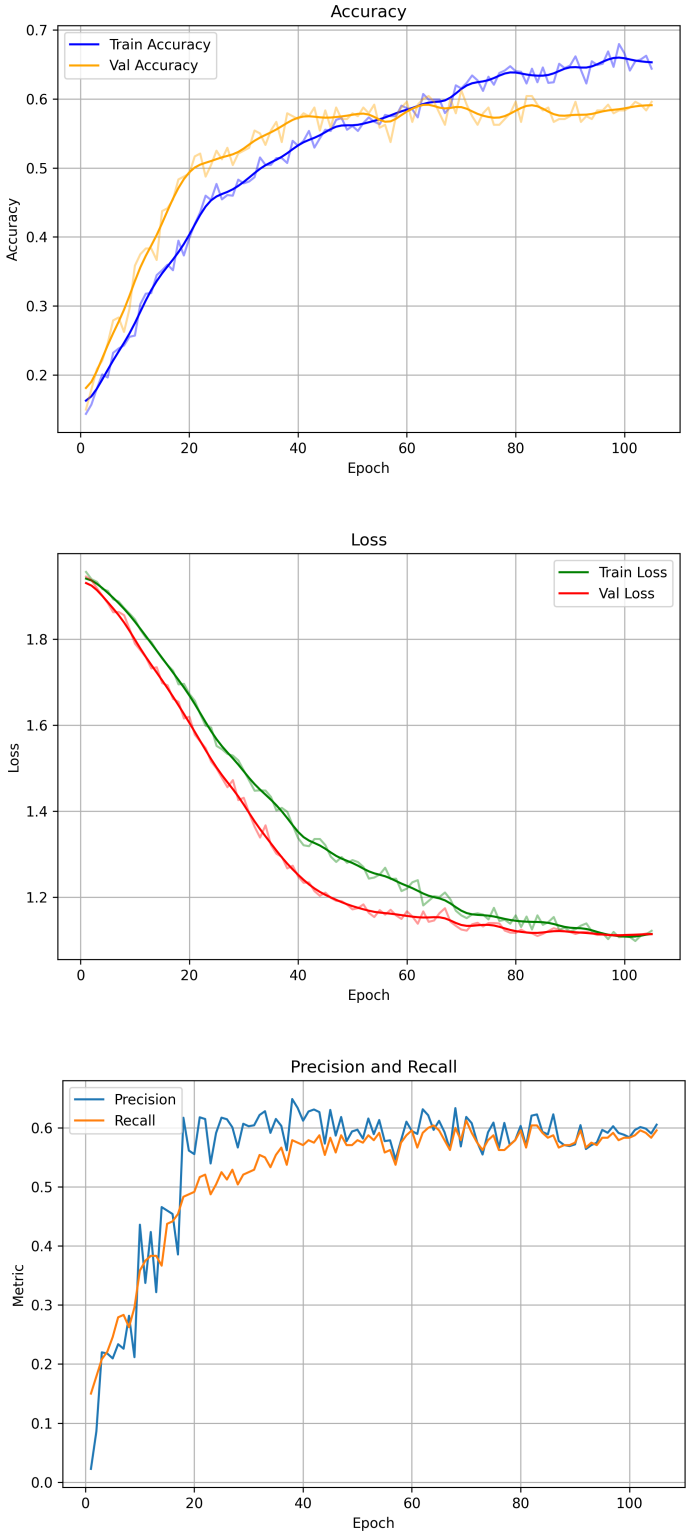
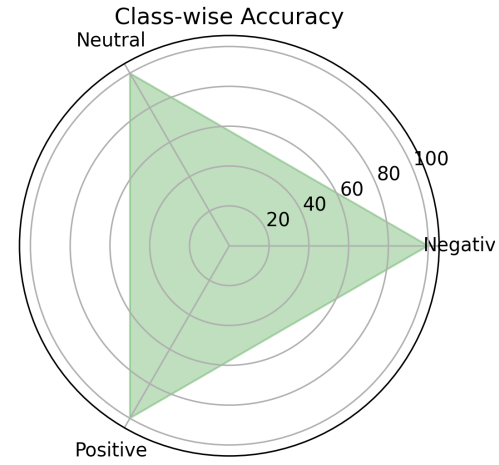
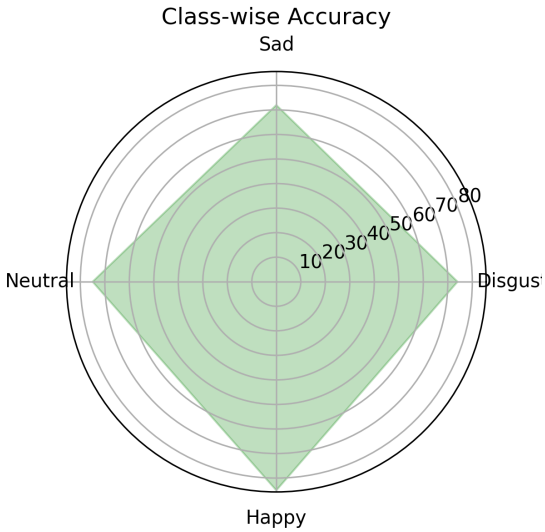


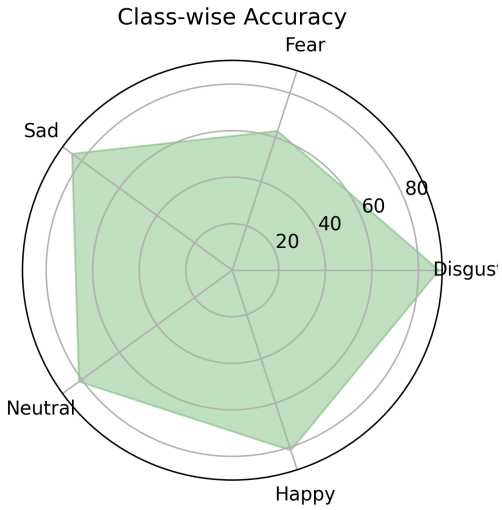
Figure 10: Learning curves for training and validation on the SEED-VII dataset



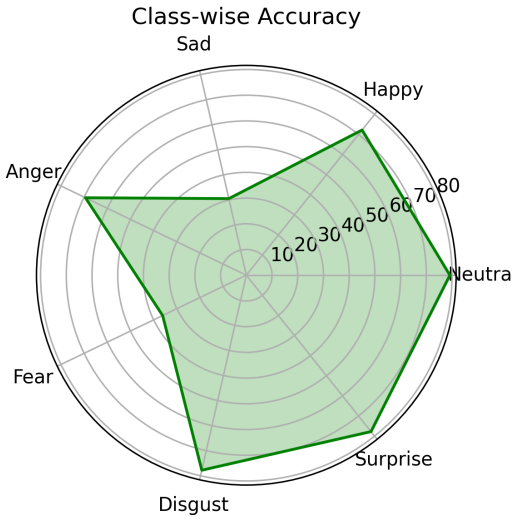
(a) Class-wise Accuracy for SEED



(b) Class-wise Accuracy for SEED-IV



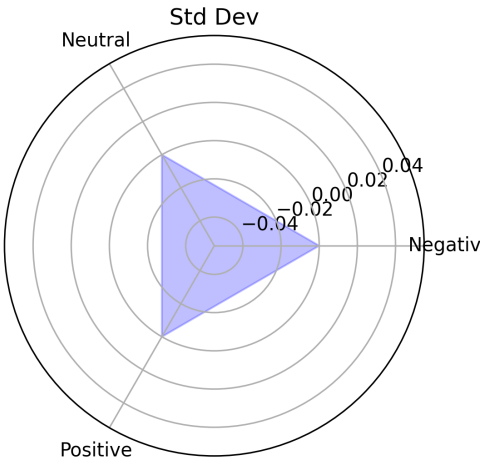
(c) Class-wise Accuracy for SEED-V



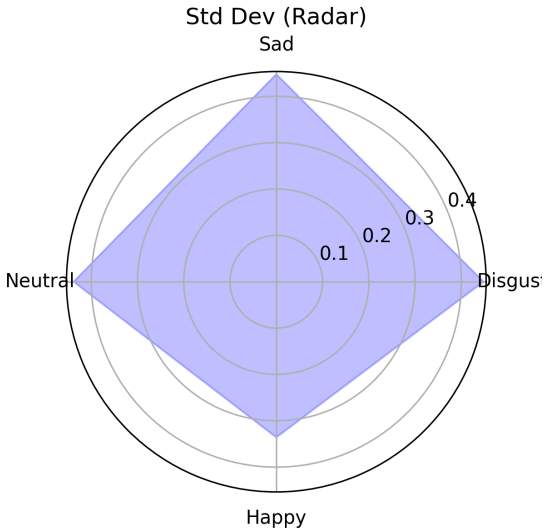
(d) Class-wise Accuracy for SEED-VII

Figure 11: Radar plots depicting per-class accuracy across multiple SEED datasets

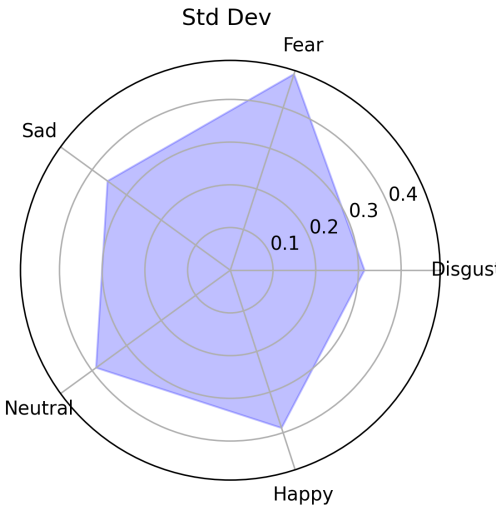
Performance variations across datasets reflect task complexity, with SEED's three emotion classes yielding higher accuracy than SEED-VII's seven, alongside potential differences in experimental protocols (e.g., stimuli, trial duration). Despite this, ResAttentioNet demonstrates strong generalization, validated through a subject-wise 70%/15%/15% split for training, validation, and testing. Radar plots of class-wise accuracy and standard deviation plots further elucidate emotion-specific performance and prediction consistency, respectively.



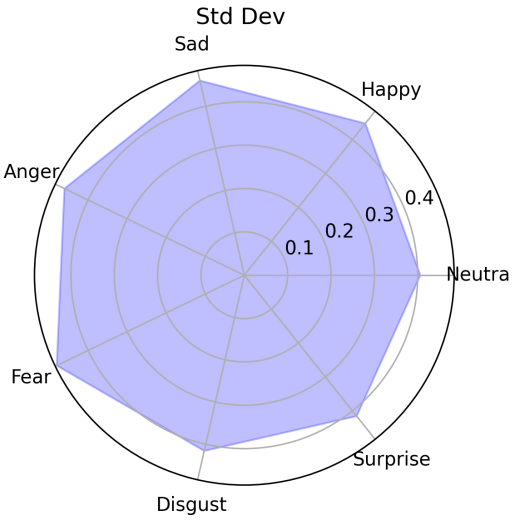
(a) Standard Deviation for SEED



(b) Standard Deviation for SEED-IV

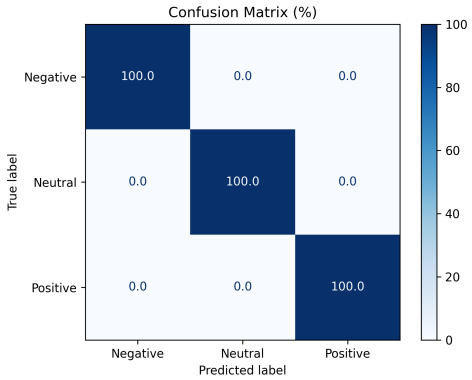


(c) Standard Deviation for SEED-V

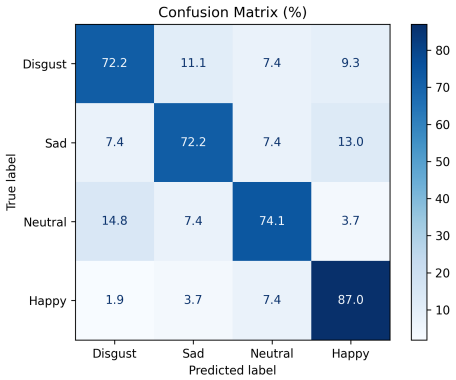


(d) Standard Deviation for SEED-VII

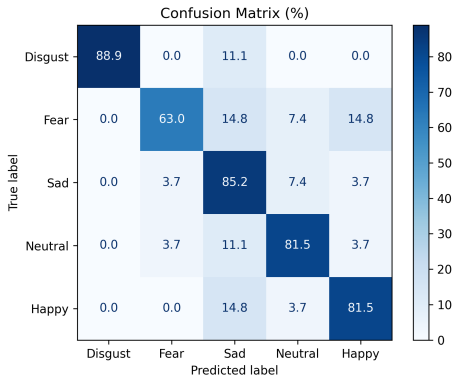
Figure 12: Radar plots depicting per-class standard deviation across multiple SEED datasets



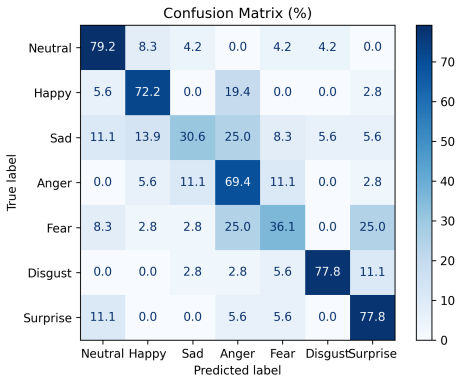
(a) Confusion Matrix for SEED



(b) Confusion Matrix for SEED-IV



(c) Confusion Matrix for SEED-V



(d) Confusion Matrix for SEED-VII

Figure 13: Visualization of performance metrics and patterns in EEG data for various models.

The original SEED dataset, with its simpler three-class taxonomy (Positive, Neutral, Negative), showed high classwise accuracy, the expanded datasets (SEED-IV, SEED-V, SEED-VII) displayed wider accuracy ranges (60–80%), with Happy and Neutral emotions consistently achieving high recognition rates. However, performance variability increased with emotional complexity—standard deviations were minimal in the SEED dataset (± 0.04) but much higher (up to 0.4) in the expanded datasets, particularly for Disgust.

Table 2: Comparison of existing models on SEED family datasets

Method	**Dataset**	**Accuracy (%)**	**F1-score (%)**
BiDANN [21]	SEED	92.38	89.06
R2G-STNN [22]	SEED	93.8	91.11
GCBNet-BLS [23]	SEED	92.24	88.84
DGCNN [24]	SEED	90.40	89.48
RGNN [25]	SEED	94.24	84.66
MSMDA [26]	SEED	89.63	93.97
GTN [27]	SEED	99.81	-
ResAttenioNet	SEED	**100**	**100**
EEGNet [28]	SEED-IV	29.89	26.59
CDCN [29]	SEED-IV	52.26	45.26
TSception [30]	SEED-IV	36.06	32.77
DGCNN [24]	SEED-IV	52.39	45.94
RGNN [25]	SEED-IV	45.40	38.24
GCBNet [23]	SEED-IV	53.28	46.26
GCBNet_BLS [23]	SEED-IV	53.51	46.91
R2G-STLT [31]	SEED-IV	75	-
DAEEGViT [32]	SEED-IV	79.45	-
ResAttenioNet	SEED-IV	**76**	**76.25**
DCCA with weighted sum fusion [33]	SEED-V	47.80	-
DCCA with attention fusion [33]	SEED-V	45.56	-
BDAE [34]	SEED-V	40.28	-
ResAttenioNet	SEED-V	**80**	**80.2**
FBCNet [35]	SEED-VII	28.32	45.56
FBCCNN [35]	SEED-VII	38.5	40.28
ResAttenioNet	SEED-VII	**62**	**61.85**

ResAttentioNet demonstrated strong and consistent performance across multiple EEG-based emotion recognition datasets. On the SEED dataset, it outperformed established models such as BiDANN, R2G-STNN, DGCNN, and RGNN in both accuracy and F1-score. For the SEED-IV dataset, it achieved higher classification performance compared to models like EEG-Net, DGCNN, and GCBNet BLS. On SEED-V, ResAttentioNet showed notable improvements over approaches such as DCCA with weighted sum fusion and BDAE. However, on the more complex SEED-VII dataset—which includes a larger number of emotional categories—its performance exhibited a relative dip when compared to its results on simpler datasets, although it remained competitive with models such as GRU, LSTM, and

FBCNet.

One major reason for this dip in performance is the imbalance between the number of classes and the corresponding growth in available training data. While the emotional category set becomes more fine-grained in datasets like SEED-VII, the dataset size does not increase proportionally to maintain an adequate number of samples per class. This reduced data-to-class ratio limits the model's ability to robustly learn distinct patterns for each emotional state, increases the risk of overfitting, and ultimately affects generalization performance in multi-class scenarios.

Conclusion and Future Scope

Our project marks a significant advancement in EEG-based emotion recognition. By employing a tailored Attention powered ResNet architecture, we have demonstrated the ability to achieve high precision in classifying emotional states from EEG data. This architecture, optimized for processing complex neural signals, showed exceptional performance, largely due to its effective extraction of salient time-frequency domain features, leading to notable improvements over baseline approaches. The robustness of our methodology, which integrates these discriminative features within a deep learning framework, is validated by our findings on the SEED family of datasets.

Our developed model not only reached better accuracy levels but also maintained consistent and reliable performance across various evaluative sessions. The result of this model underscores the potential of specialized CNNs and attention networks in deciphering the intricate, high-dimensional nature of EEG signals. However, we acknowledge challenges, particularly the limitation of working with finite datasets, which may not fully represent the diversity of real-world scenarios. The architecture's performance could be further enhanced with access to larger datasets, as deeper learning models typically excel with more data to capture complex patterns and improve generalization. Resource constraints also restricted the exploration of additional features, such as power spectral density (PSD) and connectivity features, and advanced models like Graph Neural Networks (GNNs), which could model inter-channel relationships more effectively.

The modular design of ResAttentioNet, with its distinct components, offers flexibility for future enhancements. New subcomponents can be integrated or existing ones modified to better utilize data, paving the way for targeted improvements. This study opens new avenues for future research and practical applications, especially in areas requiring nuanced understanding of human emotional states, such as advanced brain-computer interfaces and personalized affective computing.

References

[1] Wei-Long Zheng and Bao-Liang Lu. "Investigating Critical Frequency Bands and Channels for EEG-Based Emotion Recognition with Deep Neural Networks". In: *IEEE Transactions on Autonomous Mental Development* 7.3 (2015), pp. 162–175. DOI: 10.1109/TAMD.2015.2431496. URL: https://weilongzheng.github.io/publication/zheng2015investigating/zheng2015investigating.pdf.

[2] Bao-Liang Lu et al. "A Reliable Differential Entropy Feature for EEG-Based Emotion Classification". In: *IEEE Transactions on Affective Computing* 6.3 (2015), pp. 361–373. DOI: 10.1109/TAFFC.2015.2399837. URL: https://www.researchgate.net/publication/261239872_Differential_entropy_feature_for_EEG-based_emotion_classification.

[3] Yang Yang et al. "EEG-Based Emotion Recognition Using Hierarchical CNN with Spatial-Temporal Attention". In: *IEEE Access* 6 (2018), pp. 24345–24354. DOI: 10.1109/ACCESS.2018.2835502.

[4] Wei-Long Zheng and Bao-Liang Lu. "Personalizing EEG-Based Affective Models with Transfer Learning". In: *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*. 2016, pp. 2732–2738. URL: https://www.ijcai.org/Proceedings/16/Papers/388.pdf.

[5] Xiaowei Li et al. "A Novel Dual-Stream CNN for EEG-Based Emotion Recognition". In: *Sensors* 20.13 (2020), p. 3491. DOI: 10.3390/s20123491.

[6] Siddharth Tripathi et al. "Using Deep and Convolutional Neural Networks for Emotion Classification on DEAP Dataset". In: *arXiv preprint arXiv:1702.01992* (2017). URL: https://arxiv.org/abs/1702.01992.

[7] Hong Guo et al. "EEG-Based Emotion Recognition Using Spectrogram and Hybrid CNN-GRU Network". In: *IEEE Access* 10 (2022), pp. 1–10. DOI: 10.1109/ACCESS.2022.3146789.

[8] Kai Zhang et al. "Transformer-Based EEG Emotion Recognition Using Image-Like Representations". In: *Sensors* 22.7 (2022), p. 2634. DOI: 10.3390/s22072634.

[9] Hao Xu et al. "Emotion Recognition Based on Fusion of Multi-Feature Deep Learning". In: *Frontiers in Neurorobotics* 15 (2021), p. 632777. DOI: 10.3389/fnbot.2021.632777.

[10] Zheng Yin et al. "Spectral-Temporal Representation Learning for EEG-Based Emotion Recognition". In: *Neurocomputing* 415 (2021), pp. 403–412. DOI: 10.1016/j.neucom.2020.09.050.

[11] Yuan Li et al. "Cross-Session Emotion Recognition Using Multi-Channel EEG". In: *Sensors* 18.10 (2018), p. 3435. DOI: 10.3390/s18103435.

[12] Jianhua Tao et al. "Self-Supervised Representation Learning for EEG-Based Emotion Recognition". In: *IEEE Transactions on Affective Computing* 13.3 (2022), pp. 1319–1330. DOI: 10.1109/TAFFC.2020.3014879.

[13] Jian Pan et al. "Ensemble Deep Learning Models for EEG Emotion Recognition". In: *IEEE Access* 10 (2022), pp. 1–10. DOI: 10.1109/ACCESS.2022.3170092.

[14] Rui Gao et al. "Interpretable Deep Learning for EEG-Based Emotion Recognition Using Saliency Maps". In: *Neurocomputing* 483 (2022), pp. 181–190. DOI: 10.1016/j.neucom.2021.11.071.

[15] Teng Wang et al. "EEG emotion recognition based on differential entropy feature matrix through 2D-CNN-LSTM network". In: *EURASIP Journal on Advances in Signal Processing* 2024 (2024), p. 49. DOI: 10.1186/s13634-024-01146-y.

[16] Pragati Patel, Raghunandan R, and Ramesh R. "EEG-based human emotion recognition using entropy as a feature extraction measure". In: *Brain Informatics* 8.1 (2021), p. 5. DOI: 10.1186/s40708-021-00141-5.

[17] Dong-Wei Chen et al. "A Feature Extraction Method Based on Differential Entropy and Linear Discriminant Analysis for Emotion Recognition". In: *Sensors* 19.7 (2019), p. 1631. DOI: 10.3390/s19071631.

[18] Wei-Long Zheng, Jie-Yu Zhu, and Bao-Liang Lu. "Identifying stable patterns over time for emotion recognition from EEG". In: *IEEE Transactions on Affective Computing* 10.3 (2018), pp. 417–429. DOI: 10.1109/TAFFC.2017.2712143.

[19] Yuan Li, Wei-Long Zheng, and Bao-Liang Lu. "SEED-V: A multimodal dataset for continuous emotion recognition". In: *IEEE Transactions on Affective Computing* (2023). DOI: 10.1109/TAFFC.2023.3242306.

[20] Wei-Long Zheng and Bao-Liang Lu. "SEED-VII: A large-scale long-term dataset for EEG-based emotion recognition and benchmarking". In: *IEEE Transactions on Affective Computing* (2023). DOI: 10.1109/TAFFC.2023.3242310.

[21] Xianpei Li et al. "Cross-subject emotion recognition using deep domain adaptation network". In: *IEEE Transactions on Affective Computing* (2018). DOI: 10.1109/TAFFC.2018.2879499.

[22] Xianpei Li et al. "R2G-STNN: A Spatio-Temporal Neural Network for EEG Emotion Recognition". In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*. 2019.

[23] Peng Zhang et al. "GCBNet: Graph Convolutional Broad Network for EEG Emotion Recognition". In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*. 2019.

[24] Ting Song et al. "EEG Emotion Recognition Using Dynamical Graph Convolutional Neural Networks". In: *IEEE Transactions on Affective Computing* 11.3 (2020), pp. 532–541. DOI: 10.1109/TAFFC.2018.2886867.

[25] Peng Zhong, Dongrui Wang, and Changhong Miao. "EEG-based emotion recognition using regularized graph neural networks". In: *IEEE Transactions on Affective Computing* (2019). DOI: 10.1109/TAFFC.2019.2923962.

[26] Peng Zhang et al. "MSMDA: Multi-source marginal distribution adaptation for EEG-based emotion recognition". In: *IEEE Transactions on Affective Computing* (2021). DOI: 10.1109/TAFFC.2021.3059441.

[27] Murat Bilgin and Ahmet Gökhan Mert. "Gated transformer network based EEG emotion recognition". In: *Signal, Image and Video Processing* 18 (2024), pp. 6903–6910. DOI: 10.1007/s11760-024-03050-7.

[28] Vernon J Lawhern et al. "EEGNet: A compact convolutional neural network for EEG-based brain–computer interfaces". In: *Journal of Neural Engineering* 15.5 (2018), p. 056013. DOI: 10.1088/1741-2552/aace8c.

[29] Qiao Li et al. "CDCN: Channel-wise deep convolutional neural network for EEG-based emotion recognition". In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, pp. 1–8.

[30] Yijun Ding et al. "TSception: A deep learning framework for emotion detection using EEG". In: *IEEE Transactions on Affective Computing* (2020). DOI: 10.1109/TAFFC.2020.3025778.

[31] Yuhang Ouyang et al. "DAEEGViT: A domain adaptive vision transformer framework for EEG cognitive state identification". In: *Biomedical Signal Processing and Control* 100 (2025), p. 107019. DOI: 10.1016/j.bspc.2024.107019.

[32] Xiaowei Si et al. "Temporal aware Mixed Attention-based Convolution and Transformer Network for cross-subject EEG emotion recognition". In: *Computers in Biology and Medicine* 181 (2024), p. 108973. DOI: 10.1016/j.compbiomed.2024.108973.

[33] Yan Lu et al. "Deep Canonical Correlation Analysis for Multi-modal Emotion Recognition". In: *IEEE Transactions on Affective Computing* (2019). DOI: 10.1109/TAFFC.2019.2945575.

[34] Xiaobing Li et al. "A Bi-directional Deep Autoencoder for EEG-based Emotion Recognition". In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*. 2018.

[35] Yi Zhang, Hao Li, and Wei Liu. "Emotion recognition based on GRU-LSTM hybrid model from SEED-VII dataset". In: *IEEE Access* (2023). DOI: 10.1109/ACCESS.2023.3249991.

Annexure

```
import mne
import os
import json
import scipy.io as sio
import csv
import datetime
import os
import numpy as np
from typing import List, Tuple
import scipy.signal as signal
from pykalman import KalmanFilter

def extract_psd_feature(raw_data: np.array,
sample_freq: int, window_size: int, stride: int,
                        freq_bands:
        np.abs(fft_data[:, 0: int(stft_n / 2)])
        for band_index, band in enumerate(freq_bands):
            band_ave_psd = _get_average_psd(energy_graph,
            band, sample_freq, stft_n)
            psd_feature[window_index, band_index, :] = band_ave_psd
    return psd_feature

def _get_average_psd(energy_graph,
freq_bands, sample_freq,
stft_n=256):
    start_index =
    int(np.floor(freq_bands[0] /
    sample_freq * stft_n))
    end_index =
    int(np.floor(freq_bands[1] /
    sample_freq * stft_n))
    ave_psd = np.mean(energy_graph[:, start_index -
```

```
    1:end_index] ** 2, axis=1)
    return ave_psd

def get_de_feature(psd_feature: np.array):
    return np.log2(100 * psd_feature)

def smooth_feature(feature_data, method='LDS'):
    smoothed_data = np.zeros_like(feature_data)
    state_mean = np.mean(feature_data, axis=0)
    points_num, bands_num, channel_num = feature_data.shape
    for feature_band_index in range(bands_num):
        for channel_index in range(channel_num):
            kf = KalmanFilter(transition_matrices=1,
            observation_matrices=1,
            transition_covariance=0.001,
            observation_covariance=1,
            initial_state_covariance=0.1,
            initial_state_mean=
            state_mean[feature_band_index,
            channel_index])
            measurement =
            feature_data[:, feature_band_index,
            channel_index]
            smoothed_data[:, feature_band_index,
            channel_index] =
            kf.smooth(measurement)[0].flatten()
    return smoothed_data

    raw_files = os.listdir(data_path)
    file_dict = {}
    for file in raw_files:
        if file[:-15] not in file_dict.keys():
            file_dict[file[:-15]] = [file]
        else:
```

```
            file_dict[file[:-15]].append(file)
    for key, value in file_dict.items():
        EEG_preprocessed = {}
        EEG_features = {}
        for raw_file in value:
            print(raw_file)
            raw_path = os.path.join(data_path, raw_file)
            raw = mne.io.read_raw_cnt(raw_path,
            eog=['HEO', 'VEO'], ecg=['ECG'])
            raw.drop_channels(['M1', 'M2',
            'ECG', 'HEO', 'VEO'])
            montage =
            mne.channels.
            read_custom_montage(montage_file_path)
            raw.set_montage(montage)
            raw.load_data()
            raw.filter(l_freq=0.1, h_freq=70)
            raw.notch_filter(freqs=50)
            raw.resample(sfreq=200, n_jobs=4)
            trigger, _ = mne.events_from_annotations(raw)
            data, times = raw.get_data(units='uV', return_times=True)
            t = trigger[:, 0]
            if raw_file == "14_20221015_1.cnt":
                t = []
                start = datetime.
                datetime.strptime('14:25:34', '%H:%M:%S')
                with open('./SEED-VII/save_info/
                14_20221015_1_trigger_info.csv') as f:
                    trigger = csv.reader(f)
                    for row in trigger:
                        end = datetime.datetime.
                        strptime(row[1].split(' ')[-1],
                        '%H:%M:%S.%f')
                        time_diff = end.timestamp() -
```

```
                start.timestamp()
                t.append(int(round(time_diff * 200)))
    elif raw_file == "9_20221111_3.cnt":
        t = []
        start = datetime.datetime.
        strptime('14:01:27', '%H:%M:%S')
        with open('./SEED-VII/save_info
        /9_20221111_3_trigger_info.csv') as f:
            trigger = csv.reader(f)
            for row in trigger:
                end = datetime.datetime.
                strptime(row[1].split(' ')[-1], '%H:%M:%S.%f')
                time_diff = end.timestamp() - start.timestamp()
                t.append(int(round(time_diff * 200)))
    session_idx = int(raw_file[-5]) - 1
    for i in range(20):
        preprocessed_clip = data[:, t[2 * i]:t[2 * i + 1]]
        num = preprocessed_clip.shape[1] // 200
        EEG_preprocessed
        [f'{session_idx * 20 + i + 1}']
        = preprocessed_clip[:, :num * 200]
        freq_bands = [
            [1, 4],
            [4, 8],
            [8, 14],
            [14, 31],
            [31, 49]
        ]
        psd_feature =
        extract_psd_feature
        (preprocessed_clip,
        200, 4, 4, freq_bands)
        de_feature =
        get_de_feature(psd_feature)
```

```
                de_feature_smooth =
                smooth_feature(de_feature)
                EEG_features[f'psd_{session_idx * 20 + i + 1}']
                = psd_feature
                EEG_features[f'de_{session_idx * 20 + i + 1}']
                = de_feature
                EEG_features[f'de_LDS_{session_idx * 20 + i + 1}']
                = de_feature_smooth
                print(f'video num: {session_idx * 20 + i + 1},
                de shape: {de_feature.shape}')
        sio.savemat(os.path.join('./SEED-VII/EEG_features',
        key + '.mat'), EEG_features)

import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset
import numpy as np
import random
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score
import seaborn as sns
import pickle
from scipy.ndimage import gaussian_filter1d
from sklearn.utils.class_weight import compute_class_weight
from sklearn.metrics import
classification_report,
precision_recall_fscore_support, multilabel_confusion_matrix
import torch.nn.functional as F


def set_seed(seed=42):
    torch.manual_seed(seed)
```

```
    np.random.seed(seed)
    random.seed(seed)

set_seed()


X = data['X']
y = data['y']


num_samples, seq_len, num_features = X.shape
X_reshaped = X.reshape(num_samples * seq_len, num_features)


scaler = StandardScaler()
X_scaled_reshaped = scaler.fit_transform(X_reshaped)


X_scaled = X_scaled_reshaped.reshape(num_samples, seq_len, num_features


np.unique(y)


def per_subject_normalization(X, num_subjects,
trials_per_subject):
    X_norm = np.zeros_like(X)
    for subj in range(num_subjects):
        start = subj * trials_per_subject
        end = (subj + 1) * trials_per_subject
        X_subj = X[start:end]
        scaler = StandardScaler()
        X_flat = X_subj.reshape(-1, X_subj.shape[-1])
        X_scaled = scaler.fit_transform(X_flat)
```

```
        X_norm[start:end] = X_scaled.reshape(X_subj.shape)
    return X_norm

def subject_independent_split(X, y, num_subjects=16,
trials_per_subject=45, num_classes=4, train_split=0.7,
val_split=0.15, max_attempts=100):
    classes = np.unique(y)
    assert len(classes) == num_classes
    subjects = list(range(num_subjects))

    for attempt in range(max_attempts):
        np.random.shuffle(subjects)
        train_idx_end = int(train_split * num_subjects)
        val_idx_end = train_idx_end + int(val_split * num_subjects)
        train_subjects = subjects[:train_idx_end]
        val_subjects = subjects[train_idx_end:val_idx_end]
        test_subjects = subjects[val_idx_end:]

        train_idx = np.concatenate([np.arange(s *
        trials_per_subject, (s + 1) * trials_per_subject)
        for s in train_subjects])
        val_idx = np.concatenate([np.arange(s *
        trials_per_subject, (s + 1) * trials_per_subject)
        for s in val_subjects])
        test_idx = np.concatenate
        ([np.arange(s * trials_per_subject,
        (s + 1) * trials_per_subject) for s in test_subjects])

        y_train, y_val, y_test = y[train_idx], y[val_idx], y[test_idx]

        if (len(np.unique(y_train)) == num_classes and
            len(np.unique(y_val)) == num_classes and
            len(np.unique(y_test)) == num_classes):
            return X[train_idx], y_train,
```

```
            X[val_idx], y_val, X[test_idx], y_test

    raise ValueError("Could not find a valid
    split after {} attempts. Try increasing
    subjects or reducing class constraints.".format(max_attempts))

def print_class_distributions(y_train, y_val, y_test):
    print("Train classes:", np.unique(y_train, return_counts=True))
    print("Validation classes:", np.unique(y_val, return_counts=True))
    print("Test classes:", np.unique(y_test, return_counts=True))

def get_class_weights(y_train, num_classes):
    class_weights = compute_class_weight('balanced',
    classes=np.arange(num_classes), y=y_train)
    return class_weights

num_subjects, trials_per_subject, num_classes = 15,72, 4
X_normed = per_subject_normalization(X,
num_subjects, trials_per_subject)

X_train, y_train, X_val, y_val, X_test, y_test =
subject_independent_split(X_normed, y,
num_subjects, trials_per_subject,
num_classes=num_classes, train_split=0.7, val_split=0.15)
print_class_distributions(y_train, y_val, y_test)


train_dataset = TensorDataset(torch.tensor(X_train,
dtype=torch.float32), torch.tensor(y_train, dtype=torch.long))
val_dataset = TensorDataset(torch.tensor(X_val,
dtype=torch.float32), torch.tensor(y_val, dtype=torch.long))
test_dataset = TensorDataset(torch.tensor(X_test,
dtype=torch.float32), torch.tensor(y_test, dtype=torch.long))
```

```
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

class_weights = get_class_weights(y_train, num_classes)
class_weights_torch = torch.tensor(class_weights,
dtype=torch.float32).to(torch.device("cuda"
if torch.cuda.is_available() else "cpu"))

print("Class weights: ", class_weights)


class BasicBlock1D(nn.Module):
    expansion = 1
    def __init__(self, in_planes, planes,
    stride=1, downsample=None):
        super().__init__()
        self.conv1 = nn.Conv1d(in_planes, planes,
        kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm1d(planes)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv1d(planes, planes,
        kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm1d(planes)
        self.downsample = downsample

    def forward(self, x):
        identity = x
        out = self.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        if self.downsample is not None:
            identity = self.downsample(x)
        out += identity
        out = self.relu(out)
```

```
        return out

def make_layer(block, in_planes, planes, blocks, stride=1):
    downsample = None
    if stride != 1 or in_planes != planes * block.expansion:
        downsample = nn.Sequential(
            nn.Conv1d(in_planes, planes * block.expansion,
            kernel_size=1, stride=stride, bias=False),
            nn.BatchNorm1d(planes * block.expansion),
        )
    layers = []
    layers.append(block(in_planes, planes, stride, downsample))
    for _ in range(1, blocks):
        layers.append(block(planes * block.expansion, planes))
    return nn.Sequential(*layers)


class AttentionPooling(nn.Module):
    def __init__(self, embed_dim=128, num_heads=2):
        super().__init__()
        self.mha = nn.MultiheadAttention(embed_dim=embed_dim,
        num_heads=num_heads, batch_first=True)
        self.query = nn.Parameter(torch.randn(1, 1, embed_dim))
    def forward(self, x):
        batch_size = x.size(0)
        query = self.query.expand(batch_size, -1, -1)
        attn_output, _ = self.mha(query, x, x)
        return attn_output.squeeze(1)

class ResNet18_1D(nn.Module):
    def __init__(self, in_channels, base_width=32):
        super().__init__()
        self.in_planes = base_width
        self.conv1 = nn.Conv1d(in_channels, base_width,
```

```
        kernel_size=7, stride=2, padding=3, bias=False)
        self.bn1 = nn.BatchNorm1d(base_width)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool1d(kernel_size=3, stride=2, padding=1)
        self.layer1 = make_layer(BasicBlock1D,
        base_width, base_width, blocks=2)
        self.layer2 = make_layer(BasicBlock1D,
        base_width, base_width*2, blocks=2, stride=2)
        self.layer3 = make_layer(BasicBlock1D,
        base_width*2, base_width*4, blocks=2, stride=2)
        self.layer4 = make_layer(BasicBlock1D,
        base_width*4, base_width*8, blocks=2, stride=2)
        self.out_dim = base_width*8

    def forward(self, x):
        x = self.relu(self.bn1(self.conv1(x)))
        x = self.maxpool(x)
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = x.transpose(1,2)
        return x

class EmotionResNetAttention(nn.Module):
    def __init__(self, num_features=310,
    seq_len=60, num_classes=4, base_width=16):
        super().__init__()
        self.encoder = ResNet18_1D(in_channels=num_features,
        base_width=base_width)
        self.dropout = nn.Dropout(0.5)
        self.attn_pool = AttentionPooling(self.encoder.out_dim)
        self.classifier = nn.Sequential(
            nn.Linear(self.encoder.out_dim, 128),
```

```
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(128, num_classes)
        )
    def forward(self, x):
        x = x.permute(0, 2, 1)
        x = self.encoder(x)
        x = self.dropout(x)
        x = self.attn_pool(x)
        return self.classifier(x)

def plot_confusion_matrix(y_true,
y_pred, classes, filename):
    cm = confusion_matrix(y_true, y_pred)
    fig, ax = plt.subplots(figsize=(7,7))
    im = ax.imshow(cm, interpolation='nearest',
    cmap=plt.cm.Blues)
    ax.figure.colorbar(im, ax=ax)
    ax.set(xticks=np.arange(len(classes)),
           yticks=np.arange(len(classes)),
           xticklabels=classes,
           yticklabels=classes,
           title="Confusion Matrix",
           ylabel='True label',
           xlabel='Predicted label')
    plt.setp(ax.get_xticklabels(), rotation=45,
    ha="right", rotation_mode="anchor")
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], 'd'),
            ha="center", va="center",
                    color="white" if cm[i, j] >
                    thresh else "black")
```

```
    fig.tight_layout()
    plt.savefig(filename)
    plt.close()

def plot_training_summary(train_acc, val_acc,
train_loss, val_loss, TP_list, FP_list, TN_list,
FN_list, precision_list, recall_list, epoch):
    epochs_range = range(1, len(train_acc) + 1)

    smooth_train_acc = gaussian_filter1d(train_acc, sigma=2)
    smooth_val_acc = gaussian_filter1d(val_acc, sigma=2)
    smooth_train_loss = gaussian_filter1d(train_loss, sigma=2)
    smooth_val_loss = gaussian_filter1d(val_loss, sigma=2)

    plt.figure(figsize=(7,5))
    plt.plot(epochs_range,
    train_acc, color='blue', alpha=0.4)
    plt.plot(epochs_range,
    smooth_train_acc, label='Train Accuracy', color='blue')
    plt.plot(epochs_range,

    val_acc, color='orange', alpha=0.4)
    plt.plot(epochs_range,
    smooth_val_acc, label='Val Accuracy', color='orange')
    plt.title('Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.savefig(f"seedV_{epoch}_accuracy.png", dpi=300)
    plt.close()

    plt.figure(figsize=(7,5))
```

```
    plt.plot(epochs_range,
    train_loss, color='green', alpha=0.4)
    plt.plot(epochs_range,
    smooth_train_loss, label='Train Loss', color='green')
    plt.plot(epochs_range,
    val_loss, color='red', alpha=0.4)
    plt.plot(epochs_range,
    smooth_val_loss, label='Val Loss', color='red')
    plt.title('Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.savefig(f"seedV_{epoch}_loss.png", dpi=300)
    plt.close()

    plt.figure(figsize=(7,5))
    plt.plot(epochs_range, precision_list, label='Precision')
    plt.plot(epochs_range, recall_list, label='Recall')
    plt.title('Precision and Recall')
    plt.xlabel('Epoch')
    plt.ylabel('Metric')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.savefig(f"seedV_{epoch}_precision_recall.png", dpi=300)
    plt.close()

def train_eval_loop(model, train_loader,
val_loader, test_loader, epochs=100,
lr=1e-4, class_weights=None, device='cuda'):
    model = model.to(device)
```

```
optimizer = torch.optim.AdamW(model.parameters(),
lr=lr, weight_decay=1e-4)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau
(optimizer, mode='min', factor=0.5, patience=5, min_lr=1e-6)

criterion = torch.nn.CrossEntropyLoss
(label_smoothing=0.1, weight=class_weights)

best_val_acc = 0
patience, patience_counter = 10, 0

train_acc_hist, val_acc_hist = [], []
train_loss_hist, val_loss_hist = [], []
TP_list, FP_list, TN_list, FN_list = [], [], [], []
precision_list, recall_list = [], []

for epoch in range(epochs):
    model.train()
    total_loss, correct, total = 0, 0, 0
    for X_batch, y_batch in train_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)
        optimizer.zero_grad()
        output = model(X_batch)
        loss = criterion(output, y_batch)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
        correct += (output.argmax(dim=1) == y_batch).sum().item()
        total += y_batch.size(0)
    train_loss = total_loss / len(train_loader)
    train_acc = correct / total
    train_loss_hist.append(train_loss)
    train_acc_hist.append(train_acc)
```

```
model.eval()
total_loss, correct, total = 0, 0, 0
all_preds, all_labels = [], []
with torch.no_grad():
    for X_batch, y_batch in val_loader:
        X_batch, y_batch = X_batch.to(device),
        y_batch.to(device)
        output = model(X_batch)
        loss = criterion(output, y_batch)
        total_loss += loss.item()
        preds = output.argmax(dim=1)
        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(y_batch.cpu().numpy())
        correct += (preds == y_batch).sum().item()
        total += y_batch.size(0)
val_loss = total_loss / len(val_loader)
val_acc = correct / total
val_loss_hist.append(val_loss)
val_acc_hist.append(val_acc)

mcm = multilabel_confusion_matrix(all_labels,
all_preds, labels=np.unique(all_labels))
TP = sum(cm[1, 1] for cm in mcm)
FP = sum(cm[0, 1] for cm in mcm)
TN = sum(cm[0, 0] for cm in mcm)
FN = sum(cm[1, 0] for cm in mcm)
TP_list.append(TP)
FP_list.append(FP)
TN_list.append(TN)
FN_list.append(FN)

prec, rec, _, _ = precision_recall_fscore_support(all_labels, a
precision_list.append(prec)
recall_list.append(rec)
```

```
        scheduler.step(val_loss)

        print(f"Epoch {epoch+1:02d} | Train Acc:
        {train_acc:.4f} | Val Acc: {val_acc:.4f} |
        Train Loss: {train_loss:.4f} | Val Loss: {val_loss:.4f}")


plot_training_summary(train_acc_hist,
val_acc_hist, train_loss_hist, val_loss_hist,

TP_list, FP_list, TN_list, FN_list,
precision_list, recall_list, epochs)


model.eval()
total_loss, correct, total = 0, 0, 0
test_preds, test_labels = [], []
with torch.no_grad():
    for X_batch, y_batch in test_loader:
        X_batch, y_batch =
        X_batch.to(device), y_batch.to(device)
        output = model(X_batch)
        loss = criterion(output, y_batch)
        total_loss += loss.item()
        preds = output.argmax(dim=1)
        test_preds.extend(preds.cpu().numpy())
        test_labels.extend(y_batch.cpu().numpy())
        correct += (preds == y_batch).sum().item()
        total += y_batch.size(0)

test_loss = total_loss / len(test_loader)
test_acc = correct / total
precision, recall, f1, _ =
```

```
    precision_recall_fscore_support(test_labels,
    test_preds, average='weighted', zero_division=0)
    report = classification_report
    (test_labels, test_preds, zero_division=0)

    print(f"\nFinal Test Evaluation")
    print(f"Test Accuracy: {test_acc:.4f}")
    print(f"Test Loss: {test_loss:.4f}")
    print(f"Precision: {precision:.4f}
    | Recall: {recall:.4f} | F1-score: {f1:.4f}")
    print("\nClassification Report:")
    print(report)
    plot_confusion_matrix(test_labels,
    test_preds, classes=[str(i) for i in np.unique(test_labels)], filen

    return np.array(test_labels),
    np.array(test_preds),
    train_acc_hist, val_acc_hist,
    train_loss_hist, val_loss_hist,
    test_acc, test_loss, report


model = EmotionResNetAttention
(num_features=310, seq_len=60, num_classes=4)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
test_labels, test_preds,train_acc_hist,
val_acc_hist,train_loss_hist,val_loss_hist,
test_acc, test_loss, report = train_eval_loop(
    model, train_loader, val_loader,
    test_loader, epochs=100, lr=5*1e-5,
    device=device, class_weights=class_weights_torch
)
```

```
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(train_acc_hist, label='Train Accuracy')
plt.plot(val_acc_hist, label='Validation Accuracy')
plt.title("Accuracy over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(train_loss_hist, label='Train Loss')
plt.plot(val_loss_hist, label='Validation Loss')
plt.title("Loss over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.tight_layout()
plt.show()

cm = confusion_matrix(test_labels, test_preds)
cm_percent = cm.astype('float')
/ cm.sum(axis=1, keepdims=True) * 100
cm_percent_rounded = np.round(cm_percent, decimals=1)

disp = ConfusionMatrixDisplay

(confusion_matrix=cm_percent_rounded,
display_labels=['Disgust', 'Sad', 'Neutral', 'Happy'])
disp.plot(cmap='Blues', values_format='.1f')
plt.title("Confusion Matrix (%)")
plt.savefig("Confusion Matrix_seediv.png", dpi=300)
plt.show()
```

```
class_accuracies = []
class_std = []

for cls_label in np.unique(y):
    cls_indices = (test_labels == cls_label)
    if np.sum(cls_indices) > 0:
        cls_acc =
        np.mean(test_preds[cls_indices] == test_labels[cls_indices])
        class_accuracies.append(cls_acc)
        class_std.append
        (np.std(test_preds[cls_indices] == test_labels[cls_indices]))
    else:
        class_accuracies.append(0)
        class_std.append(0)


labels_radar = ['Disgust', 'Sad', 'Neutral', 'Happy']

num_labels = len(labels_radar)
class_accuracies = class_accuracies[:num_labels]
class_std = class_std[:num_labels]


class_accuracies += class_accuracies[:1]
class_std += class_std[:1]


angles = np.linspace(0, 2 * np.pi,
len(labels_radar), endpoint=False).tolist()
angles += angles[:1]

fig, ax = plt.subplots(figsize=(4, 4), subplot_kw=dict(polar=True))
ax.fill(angles, class_accuracies, alpha=0.25, color='green')
```

```
ax.set_thetagrids(np.degrees(angles[:-1]), labels_radar)
ax.set_title("Class-wise Accuracy")
plt.savefig(f"seedIV_classacc.png", dpi=300)
plt.show()


fig, ax = plt.subplots(figsize=(4, 4),
subplot_kw=dict(polar=True))
ax.fill(angles, class_std, alpha=0.25, color='blue')
ax.set_thetagrids(np.degrees(angles[:-1]), labels_radar)
ax.set_title("Std Dev (Radar)")
plt.savefig(f"seedIV_stddev.png", dpi=300)
plt.show()
```