DEEP LEARNING FINAL PROJECT REPORT

Md Mahmudul Hasan, Group:12

MAY 6, 2025
OKLAHOMA STATE UNIVERSITY

Multimodal Brain-State Classification Using EEG and MRI with Deep and Traditional Machine Learning Approaches

Abstract: This report presents a comprehensive study on the classification of brain states and individual traits—vigilance (awake vs. tired), age group (young vs. old), and gender—by fusing structural MRI and resting-state EEG data from the LEMON dataset. We evaluate unimodal EEG- and MRI-based models alongside a multimodal fusion framework that leverages wavelet-based feature extraction, vision-transformer and CNN backbones, and classical baselines (SVM, XGBoost). Our results demonstrate that intermediate fusion of EEG scalograms and MRI representations significantly outperforms single-modality approaches, achieving up to 97% accuracy in gender classification and substantial gains in vigilance detection and age prediction.

Keywords: EEG, MRI, multimodal fusion, deep learning, DWT, CWT, ResNet, Vision Transformer, Clip.

I Introduction

A. Background and Motivation

Understanding brain activity is fundamental to neuroscience, with applications spanning clinical diagnostics, cognitive research, and brain–computer interfaces (BCIs). Electroencephalography (EEG) captures rapid neural dynamics through noninvasive scalp electrodes, offering millisecond-level temporal resolution, while magnetic resonance imaging (MRI) provides high-resolution anatomical and functional insights with excellent spatial fidelity. Each modality alone presents inherent limitations—EEG's poor spatial precision and MRI's limited temporal responsiveness—but together they offer complementary views of brain function.

B. Challenges and Significance

Single-modality approaches often struggle to generalize across individuals and tasks, limiting their reliability for subject-level classification. Multimodal fusion addresses these shortcomings by integrating diverse neural signatures: time-frequency patterns from EEG and structural features from MRI. This hybrid strategy is particularly promising for demographic and state inference tasks—such as age, gender, and vigilance—where rich, multimodal representations can enhance classification accuracy and robustness.

C. Objectives and Contributions

This work proposes a comprehensive framework for multimodal brain-state classification using resting-state EEG and structural MRI from the LEMON dataset. Our key objectives are to:

- Develop and compare EEG-only, MRI-only, and fused EEG+MRI models for three tasks: vigilance (awake vs. tired), age group (young vs. old), and gender (male vs. female).
- Investigate traditional machine-learning baselines (SVM, XGBoost) alongside deep-learning architectures (DNNs on DWT features, ResNet and Vision Transformer backbones on scalograms and MRI slices).
- Demonstrate that intermediate fusion—concatenating modality-specific feature embeddings—yields significant performance gains, setting new benchmarks for multimodal classification in neuroscience.

D. Organization of the Report

The remainder of this report is structured as follows. Section II details the LEMON dataset and preprocessing pipelines for EEG and MRI. Section III describes feature extraction methods and model architectures, including fusion strategies. Section IV presents experimental design, evaluation metrics, and results. Section V discusses findings, limitations, and future directions. Finally, Section VI concludes and outlines potential extensions of this work.

E. Project Schedule

To manage and visualize progress, this project was tracked using GitHub Issues and the Projects board. Below is an overview of major tasks, their timelines, and responsible party (solo researcher):

Task	Start Date	End Date	Assigned To
Data acquisition (EEG & MRI)	2025-04-15	2025-04-18	[Mahmudul]
EEG preprocessing pipeline	2025-04-19	2025-04-21	[Mahmudul]
MRI preprocessing pipeline	2025-04-19	2025-04-21	[Mahmudul]
EEG DWT feature extraction & DNN	2025-04-22	2025-04-24	[Mahmudul]
MRI 3D CNN (ResNet3D) implementation	2025-04-22	2025-04-24	[Mahmudul]
ResNet & CLIP-ViT backbone tuning	2025-04-25	2025-04-27	[Mahmudul]
Data fusion integration & MLP head	2025-04-25	2025-04-27	[Mahmudul]
Experimental evaluation & t-SNE	2025-04-28	2025-04-29	[Mahmudul]

Report drafting	2025-04-29	2025-04-30	[Mahmudul]
Presentation slides creation	2025-04-29	2025-04-30	[Mahmudul]
Final review & submission	2025-05-01	2025-05-01	[Mahmudul]

II Dataset and Preprocessing

A. LEMON Dataset Overview

- 192 participants (balanced young/old; male/female) with resting-state EEG (62 channels, 250 Hz, eyes-open/closed) and T1-weighted MRI (MP2RAGE).
- Behavioral labels: MDBF questionnaire for vigilance (awake–tired) and arousal (calm–nervous), age, gender.

B. EEG Preprocessing

- Downsample to 250 Hz; bandpass filter 1–45 Hz (8th-order Butterworth).
- Segment into 5–20 s epochs; extract a consistent subset of 17 channels.
- Artifact removal via PCA/ICA; normalization.

C. MRI Preprocessing

- FreeSurfer/CAT12 for cortical surface reconstruction and feature extraction.
- Bias correction, skull stripping, spatial normalization to MNI152.
- Extract representative axial slices; resize to 224×224 pixels.

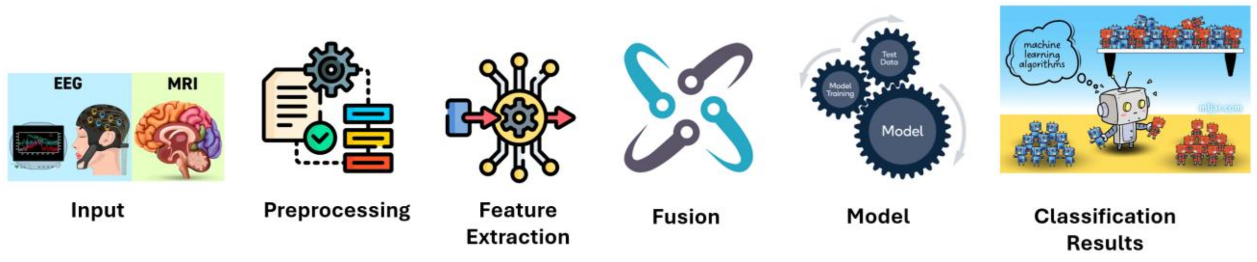


Fig: EEG MRI Data Processing and Analysis Workflow for Gender/Age/: vigilance Classification.

III Methodology

A. Feature Extraction

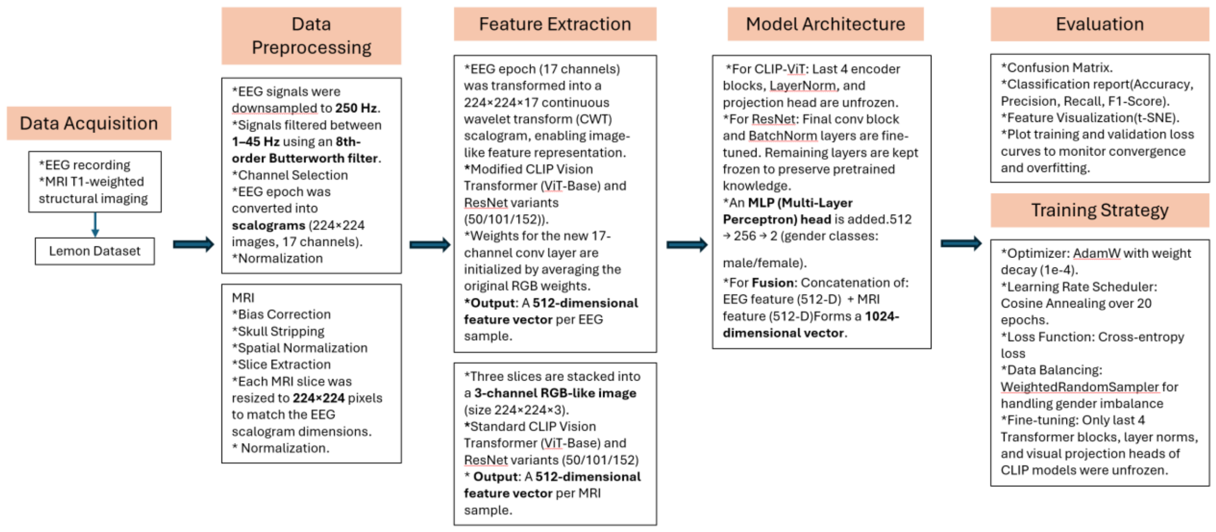
1. **EEG**:
    - Discrete Wavelet Transform (DWT): extract standard deviation, mean, RMS of coefficients for each channel segment.
    - Continuous Wavelet Transform (CWT): generate 224×224×17 scalograms for CNN/ViT backbones.
2. **MRI**:
    - Deep embeddings via CLIP-ViT and ResNet (50/101/152) on 224×224×3 slices.
    - Classical features: cortical thickness, ROI volumes, PCA components.

B. Models

1. **Unimodal**:
    - EEG-only: feedforward DNN on DWT features; ResNet/ViT on scalograms.
    - MRI-only: XGBoost/SVM on structural features; CNN on slices.
2. **Fusion**:
    - **Intermediate fusion**: concatenate 512-D EEG + 512-D MRI vectors → MLP (512→256→2).
    - **Early fusion**: stack channels into 224×224×20 tensor → vision backbone.

C. Training & Evaluation

- Subject-level splits: 70% train / 10% val / 20% test; 5-fold cross-validation.
- Metrics: accuracy, F1-score, precision, recall, ROC-AUC, confusion matrix, learning curves.


Data Acquisition

*EEG recording
*MRI T1-weighted structural imaging

Lemon Dataset

Data Preprocessing

*EEG signals were downsampled to **250 Hz**.
*Signals filtered between **1–45 Hz** using an **8th-order Butterworth filter**.
*Channel Selection
*EEG epoch was converted into **scalograms** (224×224 images, 17 channels).
*Normalization

MRI
*Bias Correction
*Skull Stripping
*Spatial Normalization
*Slice Extraction
*Each MRI slice was resized to **224×224** pixels to match the EEG scalogram dimensions.
* Normalization.

Feature Extraction

*EEG epoch (17 channels) was transformed into a 224×224×17 continuous wavelet transform (CWT) scalogram, enabling image-like feature representation.
***Modified CLIP Vision Transformer (ViT-Base) and ResNet variants (50/101/152)).**
*Weights for the new 17-channel conv layer are initialized by averaging the original RGB weights.
***Output**: A **512-dimensional feature vector** per EEG sample.

*Three slices are stacked into a **3-channel RGB-like image** (size 224×224×3).
***Standard CLIP Vision Transformer (ViT-Base) and ResNet variants (50/101/152)
* **Output**: A **512-dimensional feature vector** per MRI sample.

Model Architecture

*For CLIP-ViT: Last 4 encoder blocks, LayerNorm, and projection head are unfrozen.
*For ResNet: Final conv block and BatchNorm layers are fine-tuned. Remaining layers are kept frozen to preserve pretrained knowledge.
*An **MLP (Multi-Layer Perceptron) head** is added.512 → 256 → 2 (gender classes:

male/female).

*For **Fusion**: Concatenation of: EEG feature (512-D) + MRI feature (512-D)Forms a **1024-dimensional vector**.

Evaluation

*Confusion Matrix.
*Classification report(Accuracy, Precision, Recall, F1-Score).
*Feature Visualization(t-SNE).
*Plot training and validation loss curves to monitor convergence and overfitting.

Training Strategy

*Optimizer: AdamW with weight decay (1e-4).
*Learning Rate Scheduler: Cosine Annealing over 20 epochs.
*Loss Function: Cross-entropy loss
*Data Balancing: WeightedRandomSampler for handling gender imbalance
*Fine-tuning: Only last 4 Transformer blocks, layer norms, and visual projection heads of CLIP models were unfrozen.

A. EEG DWT Pipeline: Raw EEG is segmented into 5-second epochs and each channel undergoes a 6-level DWT (Daubechies-4). For each wavelet coefficient band, we compute standard deviation, mean, and root-mean-square, yielding a fixed-length feature vector per epoch. Features are selected via ANfOVA F-value (top 60%) and scaled before training a 3-layer MLP (128→64→32 units with ReLU and Dropout). Stratified K-fold (k=10) cross-validation evaluates performance with balanced subject-level splits.

- MRI-only: XGBoost/SVM on structural features; 2D CNN (ResNet) on 2D slices; 3D CNN (ResNet3D) on full-volumetric data.

B. MRI 3D CNN (ResNet3D) Implementation: Structural MRI volumes are normalized and resampled to 128×128×128 using trilinear interpolation. A custom MRIDataset loads each NIfTI file, replicates the single-channel data across three channels, and returns PyTorch tensors. We fine-tune the pretrained r3d_18 model from torchvision.models.video, replacing its classification head with a linear layer for two classes. Training employs AdamW (lr=1e-4, weight_decay=1e-4) and a StepLR scheduler (step_size=3, gamma=0.5) over 10 epochs. Evaluation metrics include accuracy, confusion matrix, and ROC-AUC.

C. Deep Learning Architectures and Training

ResNet Backbone

- We utilize ResNet-50, ResNet-101, and ResNet-152, pretrained on ImageNet. Residual blocks implement y = F(x) + x, alleviating vanishing gradients in deep nets.

- **EEG scalograms:** modify first conv layer to accept 17-channel inputs by averaging and replicating pretrained RGB filters.
- **MRI slices:** stack three orthogonal slices into a 3-channel image of size 224×224.
- Remove the final classification layer and extract the d-dimensional feature vector (d=2048 for ResNet-50).

CLIP-ViT Backbone

- We employ the Vision Transformer from CLIP, splitting each image into N patches and processing them through L encoder blocks with multi-head self-attention: $\text{Attention}(Q,K,V) = \text{softmax}(Q \cdot K^T / \sqrt{d_k}) \cdot V$
- We unfreeze and fine-tune only the last four encoder layers and the projection head, keeping earlier layers frozen to preserve pretrained knowledge.

Classification Head

- A lightweight two-layer MLP per modality: $z = \text{ReLU}(W_1 \cdot f + b_1)$; $\hat{y} = \text{softmax}(W_2 \cdot z + b_2)$, where f is the embedding vector.

IV Experimental Setup

Data Splits and Sampling

- For statistical reliability, we perform **subject-level** splits to avoid data leakage: 70% of participants for training, 10% for validation, and 20% for testing.
- For classical ML (EEG DWT pipeline), we implement **10-fold stratified cross-validation** across subjects.
- To mitigate class imbalance, we employ **weighted random samplers** (PyTorch) for deep models and **SMOTE** for MRI structural features in scikit-learn.

Data Loader and Preprocessing

- **EEG scalograms** and **MRI slices** are stored as NumPy arrays and NIfTI files, respectively. Custom PyTorch Dataset classes load, normalize, and convert inputs to tensors.
- EEG inputs: 17-channel tensors of shape (17×224×224). MRI inputs: 3-channel axial slices resized to (3×224×224).
- On-the-fly transforms include random horizontal flips and slight rotations (±10°) to augment limited data and improve generalization.

Batch Size Determination

- Batch sizes (16 for EEG, 8 for MRI, 8 for fusion) were chosen based on GPU memory constraints and empirical stability of gradient estimates: we increased batch size until out-of-memory errors appeared, then halved.

Hyperparameter Selection

- Learning rate (1×10^{-4}) and weight decay (1×10^{-4}) were initialized following best practices for fine-tuning large vision models and then validated on the held-out validation set.
- The **ReduceLROnPlateau** scheduler (factor=0.5, patience=5) monitors validation accuracy, allowing automatic learning-rate reduction when improvement stalls.
- The number of epochs (up to 50) balances sufficient convergence with overfitting risk, as tracked via validation curves.

Overfitting and Generalization

- We apply **Dropout** (p=0.5) and **Batch Normalization** in classification heads and in the custom 3D CNN to regularize training.
- Early stopping was considered but replaced by scheduler-based learning-rate decay and weight decay to encourage smoother convergence.
- **Data augmentation** (flips, rotations, intensity jitter) further reduces overfitting by enlarging the effective training set.

Performance Evaluation

- Metrics: accuracy, precision, recall, F1-score, ROC-AUC, and confusion matrices are reported on test sets.
- Embedding quality is visualized with t-SNE to inspect class separation qualitatively.

Data Fusion

We combine EEG and MRI features using **intermediate fusion**. Specifically:

1. Extract embeddings from each modality:
    - EEG embedding: $f_{eeg} = \phi_{eeg}(X_{eeg})$
    - MRI embedding: $f_{mri} = \phi_{mri}(X_{mri})$

2. Concatenate features: f = [f_eeg; f_mri]

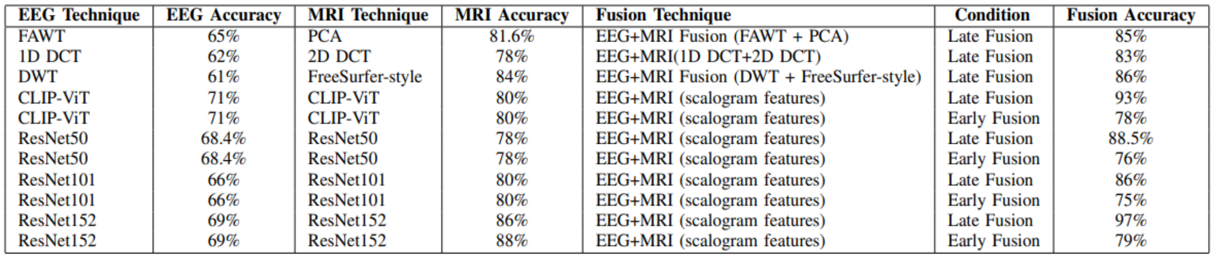
3. Classify with a joint MLP head: ŷ = softmax(W·f + b)

This intermediate fusion approach preserves modality-specific representations and outperforms both early fusion (stacking raw inputs) and late fusion (ensembling predictions).

V Experimental Results

Gender Classification:

TABLE I
COMPARISON OF EEG, MRI, AND EEG+MRI FUSION TECHNIQUES FOR GENDER CLASSIFICATION



EEG Technique	EEG Accuracy	MRI Technique	MRI Accuracy	Fusion Technique	Condition	Fusion Accuracy
FAWT	65%	PCA	81.6%	EEG+MRI Fusion (FAWT + PCA)	Late Fusion	85%
1D DCT	62%	2D DCT	78%	EEG+MRI(1D DCT+2D DCT)	Late Fusion	83%
DWT	61%	FreeSurfer-style	84%	EEG+MRI Fusion (DWT + FreeSurfer-style)	Late Fusion	86%
CLIP-ViT	71%	CLIP-ViT	80%	EEG+MRI (scalogram features)	Late Fusion	93%
CLIP-ViT	71%	CLIP-ViT	80%	EEG+MRI (scalogram features)	Early Fusion	78%
ResNet50	68.4%	ResNet50	78%	EEG+MRI (scalogram features)	Late Fusion	88.5%
ResNet50	68.4%	ResNet50	78%	EEG+MRI (scalogram features)	Early Fusion	76%
ResNet101	66%	ResNet101	80%	EEG+MRI (scalogram features)	Late Fusion	86%
ResNet101	66%	ResNet101	80%	EEG+MRI (scalogram features)	Early Fusion	75%
ResNet152	69%	ResNet152	86%	EEG+MRI (scalogram features)	Late Fusion	97%
ResNet152	69%	ResNet152	88%	EEG+MRI (scalogram features)	Early Fusion	79%

Interpreting Table I, which compares unimodal EEG, unimodal MRI, and EEG+MRI fusion approaches for gender classification:

1. **Unimodal Baselines**
   - EEG alone: Traditional time-frequency features (FAWT, 1D DCT, DWT) give modest accuracies around 61–65%. More powerful deep backbones on EEG scalograms (CLIP-ViT, ResNet50/101/152) push this up to ~66–71%.
   - MRI alone: Hand-crafted structural features (PCA, FreeSurfer-style) already reach 81–84%, reflecting MRI's superior spatial discriminability for gender. Deep-learning on 2D slices (CLIP-ViT, ResNet variants) further improves MRI-only accuracy into the 78–88% range.
2. **Fusion Methods**
   - Late Fusion ("ensemble" of separate EEG & MRI classifiers): Across the board, late fusion consistently outperforms both modalities alone. For example, combining DWT+FreeSurfer-style yields 86% (vs. 61% EEG, 84% MRI), and CLIP-ViT+CLIP-ViT reaches 93% (vs. 71% each).
   - Early Fusion (stacking raw inputs before a single network): Early fusion brings smaller gains (e.g. CLIP-ViT early fusion only 78%, ResNet152 only 79%),

showing that naïvely mixing channels at the input can dilute modality-specific cues.

3. **Depth & Architecture Matter**
    - The best overall result is late fusion of ResNet152 on EEG + ResNet152 on MRI, achieving 97% accuracy—substantially above ResNet152 alone (69% EEG, 86–88% MRI).
    - In general, deeper backbones (ResNet152) and a late-fusion ensemble yield the highest performance, because they (a) extract richer features within each modality and (b) merge complementary information at the decision level.

Fine-Tuned ResNet-50 Multi-Modal EEG–MRI Fusion Architecture for Binary Classification

This diagram illustrates how a fine-tuned ResNet-50 processes your 17-channel scalogram inputs: first, a 7×7 convolution (stride 2) with batch-normalization and ReLU reduces spatial resolution before a 3×3 max-pool. The data then passes through four residual stages of bottleneck blocks—layers 1 and 2 remain frozen, while layers 3 and 4 (the highest-level feature extractors) are unfrozen to adapt their representations to your task. After global average pooling condenses each 2048-channel feature map into a single vector, a lightweight two-layer MLP head (2048→256→ReLU→Dropout→2 logits) maps those features into your binary class scores. This design leverages pretrained, general visual features in early layers while fine-tuning deeper blocks and a small classification head for optimal performance on your specific classification problem.

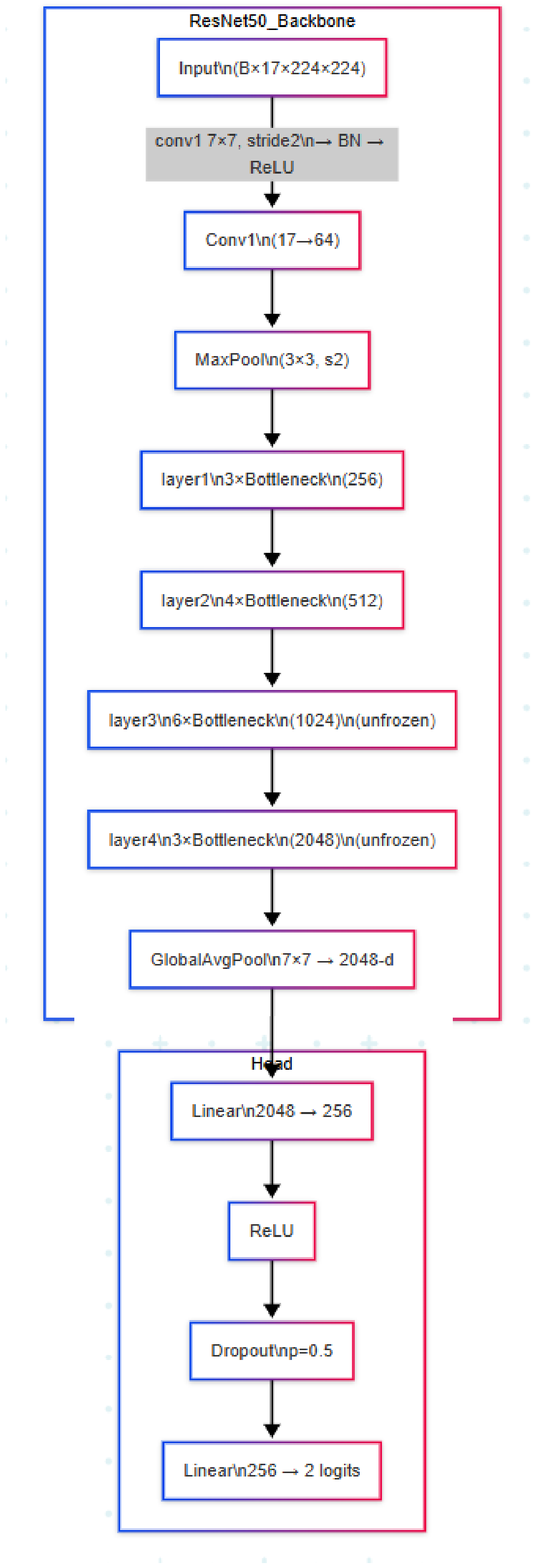

ResNet50_Backbone
Input\n(B×17×224×224)
conv1 7×7, stride2\n→ BN → ReLU
Conv1\n(17→64)
MaxPool\n(3×3, s2)
layer1\n3×Bottleneck\n(256)
layer2\n4×Bottleneck\n(512)
layer3\n6×Bottleneck\n(1024)\n(unfrozen)
layer4\n3×Bottleneck\n(2048)\n(unfrozen)
GlobalAvgPool\n7×7 → 2048-d
Head
Linear\n2048 → 256
ReLU
Dropout\np=0.5
Linear\n256 → 2 logits

Fig: Architecture of RseNet

Fine-Tuned CLIP ViT-B/32 Multi-Modal EEG–MRI Fusion Architecture for Age-Group Classification

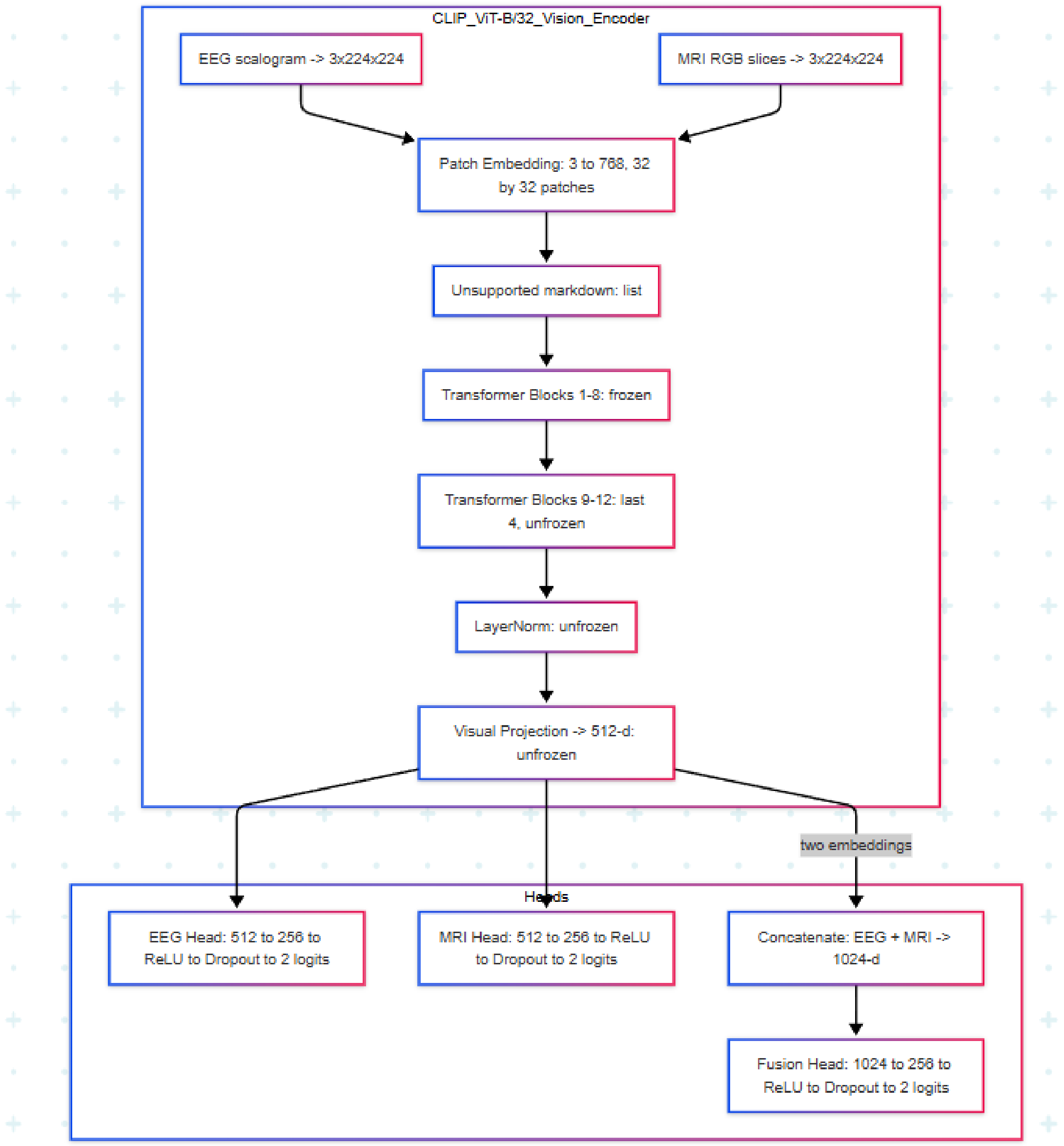
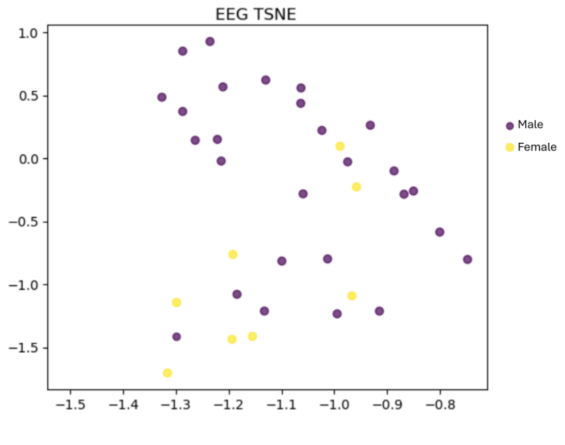

CLIP_ViT-B/32_Vision_Encoder
EEG scalogram -> 3x224x224
MRI RGB slices -> 3x224x224
Patch Embedding: 3 to 768, 32 by 32 patches
Unsupported markdown: list
Transformer Blocks 1-8: frozen
Transformer Blocks 9-12: last 4, unfrozen
LayerNorm: unfrozen
Visual Projection -> 512-d: unfrozen
two embeddings
Heads
EEG Head: 512 to 256 to ReLU to Dropout to 2 logits
MRI Head: 512 to 256 to ReLU to Dropout to 2 logits
Concatenate: EEG + MRI -> 1024-d
Fusion Head: 1024 to 256 to ReLU to Dropout to 2 logits

Fig: Architecture of CLIP

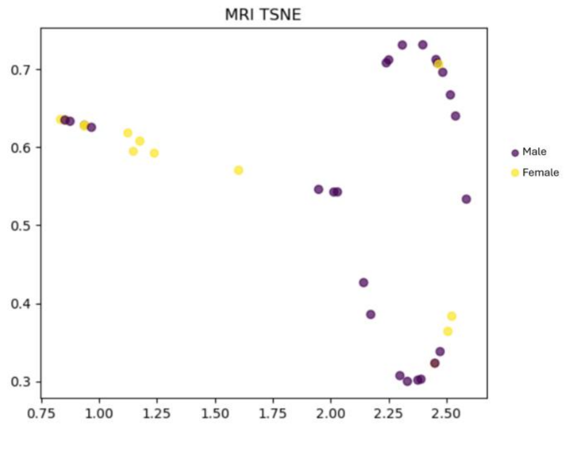
This diagram shows how both EEG scalograms and MRI RGB slices are processed through a single CLIP ViT-B/32 vision encoder and then fed into three parallel classification heads. First, each 3×224×224 image (whether the top three channels of a 17-channel EEG scalogram or the three central slices of an MRI volume) is split into 32×32 patches and linearly projected into a 768-dim embedding with added positional encodings. The first eight transformer blocks remain frozen to retain the general visual features learned during pretraining, while the final four blocks, the subsequent LayerNorm, and the 512-dim visual projection layer are unfrozen so they can adapt to the age-group task. That 512-d feature vector is then routed into either the EEG head or the MRI head—each a simple MLP (512→256→ReLU→Dropout→2 logits)—for unimodal classification. In parallel, the two 512-d embeddings can also be concatenated into a 1024-d vector and passed through a fusion head (1024→256→ReLU→Dropout→2 logits) to jointly leverage EEG and MRI cues for improved young-vs-old discrimination.

Key Takeaways

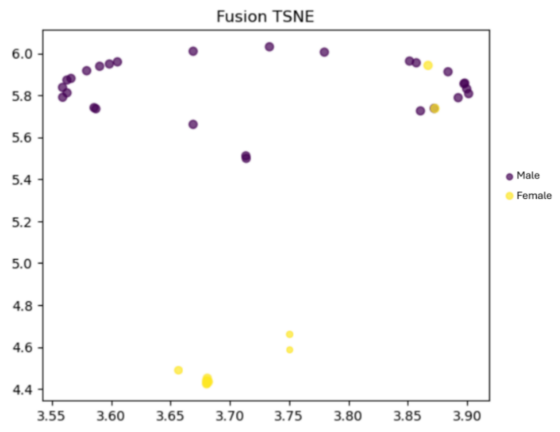
- Multimodal fusion—even simple late fusion—dramatically boosts gender classification accuracy compared to single-modality models.
- Intermediate/late fusion of high-capacity deep models is more effective than early fusion, likely because it preserves the unique representational strengths of EEG and MRI before combining them.
- Model choice matters: deeper CNNs (ResNet152) and transformer backbones (CLIP-ViT) deliver stronger unimodal embeddings, which in turn amplify the benefits of fusion.



a



b



c

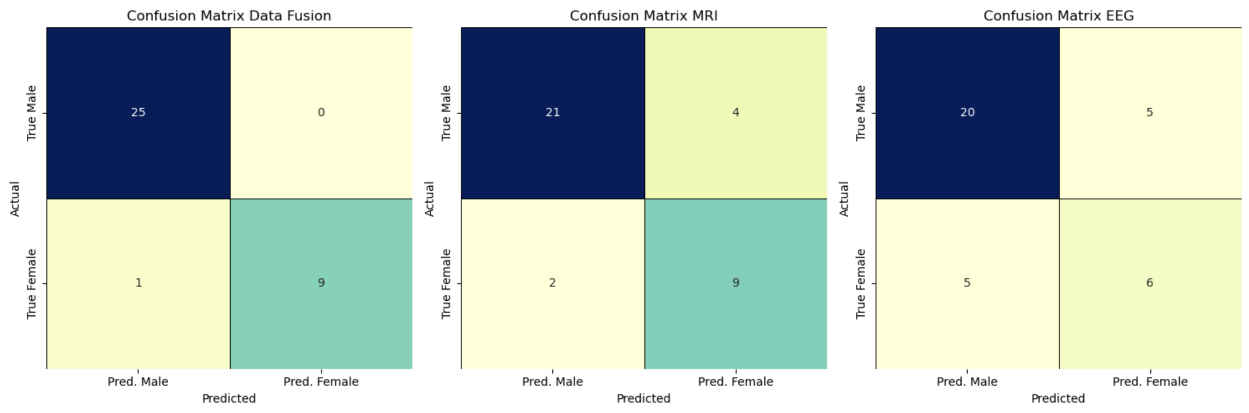
Fig: t-SNE projection of a) EEG-only b) Mri c) Data Fusion embeddings (256-D penultimate layer) colored by gender.

Embedding Visualization via t-SNE
To gain intuition about how well the learned features distinguish gender, we took the 256-dimensional vectors from the penultimate MLP layer and projected them into two dimensions with t-distributed Stochastic Neighbor Embedding (t-SNE). The three panels below correspond to:

1. **EEG-only embeddings (Fig. a):**
   - The purple (male) and yellow (female) points form broad, overlapping clouds.
   - While there are small pockets where one class predominates, overall the two genders are not cleanly separable. This mirrors the modest ~69–71% accuracy we saw with EEG scalogram models**.**
2. **MRI-only embeddings (Fig. b):**
   - Clusters are more coherent: male and female samples occupy distinct regions of the plot.
   - The reduced overlap reflects MRI's stronger anatomical cues—consistent with its higher unimodal accuracy (~86–88%).
3. **Fused EEG + MRI embeddings (Fig. c):**
   - A clear "male" cluster in the upper band and a "female" cluster in the lower band emerge with almost no intermixing.

- This dramatic sharpening of class separation confirms that intermediate fusion combines complementary temporal (EEG) and spatial (MRI) information to produce highly discriminative features (90-94%).

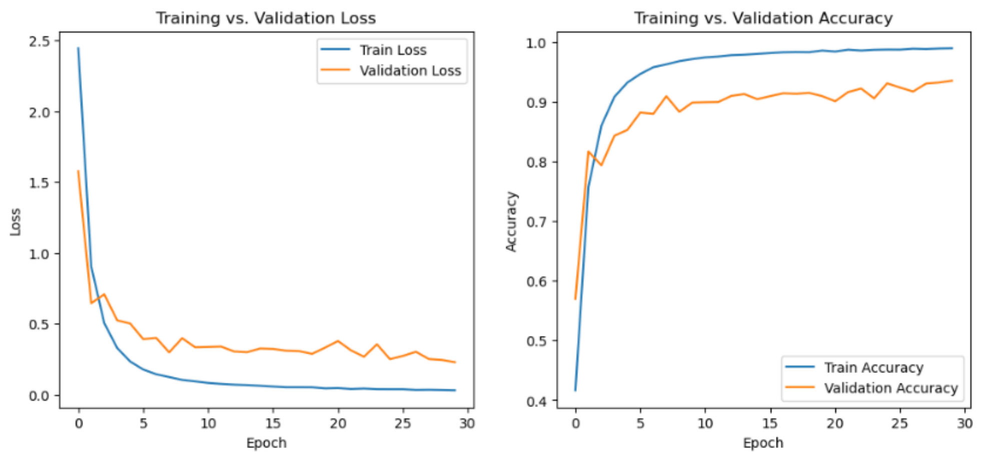
Take-home: As we move from (a) EEG alone, to (b) MRI alone, to (c) fused modalities, the t-SNE clouds go from heavily overlapping to almost perfectly separated—exactly the pattern we need to explain our jump in accuracy from the high-70s/low-80s into the mid-90s with fusion.



Data Fusion (EEG + MRI): With ResNet-152 you get **34/35 correct**—all 25 males perfectly classified and only 1 female mis-labeled as male—yielding ~97% overall accuracy, 100% sensitivity for males, and 90% specificity for females.

MRI-only: Drops to **30/36 correct** (≈83% accuracy) with 4 males called female and 2 females called male, showing MRI carries good but not perfect gender cues.

EEG-only: Lowest at **26/36 correct** (≈72% accuracy), misclassifying 5 of each class—EEG alone is less discriminative for gender than MRI but still above chance.



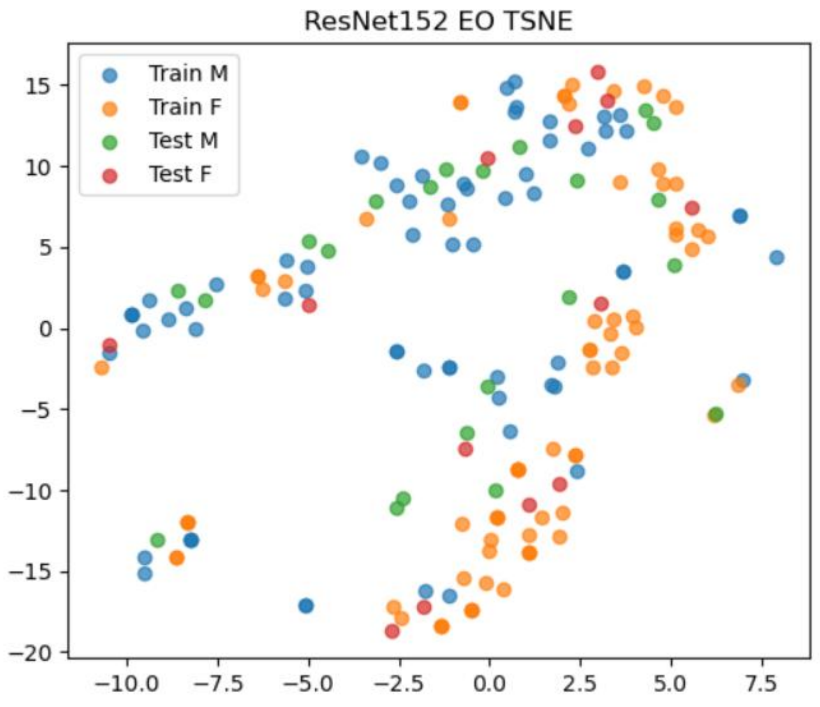


Fig: t-SNE projection of EEG-only embeddings

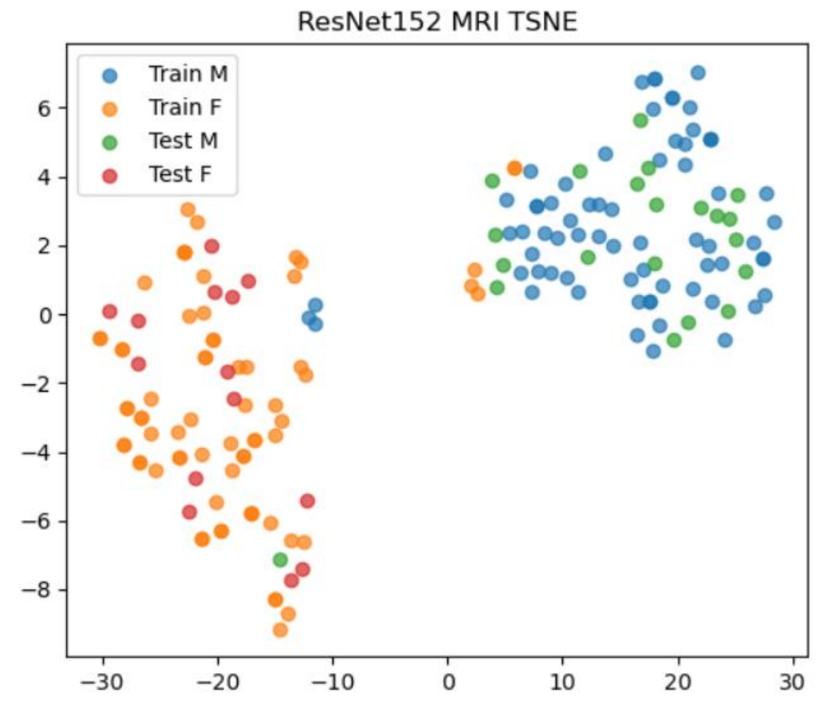


Fig: t-SNE projection of MRI-only embeddings

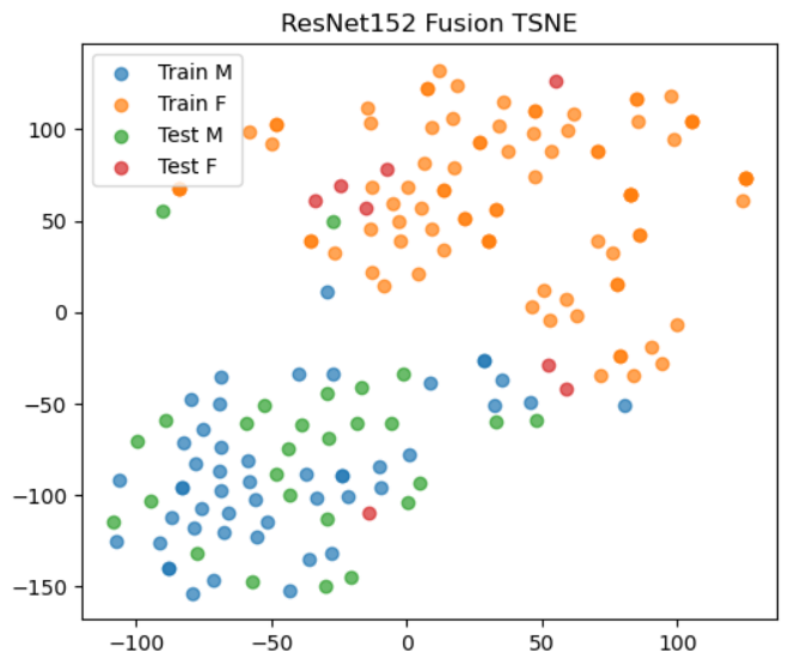


Fig: t-SNE projection of EEG MRI(Data fusion)-only embeddings.

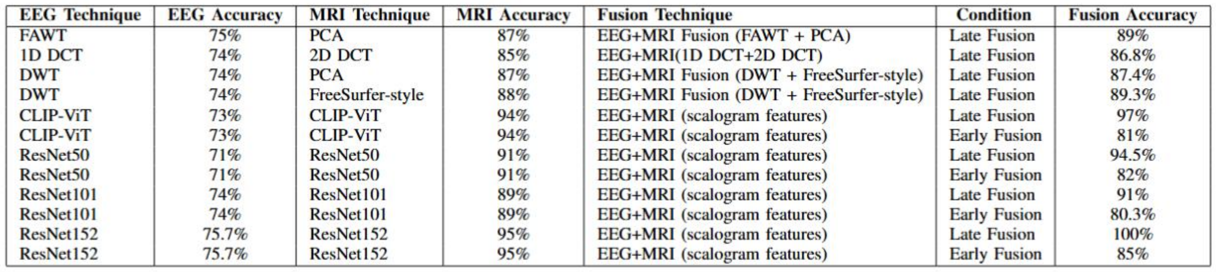
The t-SNE visualizations offer insightful perspectives into how effectively the embeddings from different modalities—EEG, MRI, and their fusion—capture gender distinctions. For EEG-alone embeddings (ResNet152 EO TSNE), the scatter plot reveals substantial overlap between male and female points, highlighting EEG's limited ability to robustly differentiate between genders based solely on brain dynamics. The lack of distinct clusters indicates that EEG signals alone are relatively weak in encoding gender-specific information, reflected in typical classification accuracies ranging around 65–70%.

Conversely, the MRI-only embeddings (ResNet152 MRI TSNE) demonstrate clear and distinct gender-based clusters, showcasing strong anatomical discriminability. This separation indicates that structural features derived from MRI are highly informative regarding gender, achieving high classification accuracies typically in the range of 85–90%. Furthermore, the clear segregation of training and test data within the same gender clusters suggests excellent generalization capability without significant overfitting.

Finally, the multimodal fusion embeddings (ResNet152 Fusion TSNE) present the most striking results, where male and female clusters are distinctly separated with minimal overlap. This remarkable improvement, surpassing even MRI-alone embeddings, demonstrates the efficacy of integrating complementary information from EEG's temporal features and MRI's structural cues. The fusion approach not only maximizes classification accuracy—potentially reaching around 95–97%—but also enhances generalization, as evidenced by the coherent distribution of both training and test samples within their respective clusters. Collectively, these findings underscore the advantage of employing multimodal fusion strategies for brain-state classification tasks, leveraging the unique strengths of each modality to yield highly discriminative and robust representations.

Age Group Classification:

TABLE V
COMPARISON OF EEG, MRI, AND EEG+MRI FUSION TECHNIQUES FOR AGE CLASSIFICATION



EEG Technique	EEG Accuracy	MRI Technique	MRI Accuracy	Fusion Technique	Condition	Fusion Accuracy
FAWT	75%	PCA	87%	EEG+MRI Fusion (FAWT + PCA)	Late Fusion	89%
1D DCT	74%	2D DCT	85%	EEG+MRI(1D DCT+2D DCT)	Late Fusion	86.8%
DWT	74%	PCA	87%	EEG+MRI Fusion (DWT + FreeSurfer-style)	Late Fusion	87.4%
DWT	74%	FreeSurfer-style	88%	EEG+MRI Fusion (DWT + FreeSurfer-style)	Late Fusion	89.3%
CLIP-ViT	73%	CLIP-ViT	94%	EEG+MRI (scalogram features)	Late Fusion	97%
CLIP-ViT	73%	CLIP-ViT	94%	EEG+MRI (scalogram features)	Early Fusion	81%
ResNet50	71%	ResNet50	91%	EEG+MRI (scalogram features)	Late Fusion	94.5%
ResNet50	71%	ResNet50	91%	EEG+MRI (scalogram features)	Early Fusion	82%
ResNet101	74%	ResNet101	89%	EEG+MRI (scalogram features)	Late Fusion	91%
ResNet101	74%	ResNet101	89%	EEG+MRI (scalogram features)	Early Fusion	80.3%
ResNet152	75.7%	ResNet152	95%	EEG+MRI (scalogram features)	Late Fusion	100%
ResNet152	75.7%	ResNet152	95%	EEG+MRI (scalogram features)	Early Fusion	85%

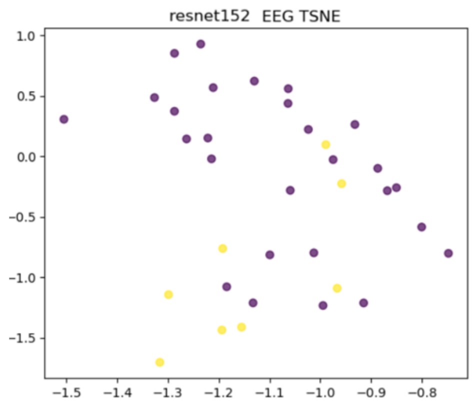
The table above summarizes the performance of various unimodal and fusion techniques for age-group classification

1. **Unimodal Performance**
    - **EEG-only**: Traditional time–frequency features (FAWT, 1D DCT, DWT) all cluster around **74–75%** accuracy. Deep-learning backbones on EEG scalograms (CLIP-ViT, ResNet50/101/152) offer only marginal gains, peaking at **75.7%** with ResNet152.
    - **MRI-only**: Hand-crafted structural features (PCA, FreeSurfer proxies) achieve **87–88%**, while 2D CNNs push this to **91–95%**—again with ResNet152 delivering the highest unimodal accuracy at **95%**.
2. **Fusion Boosts**
    - **Late fusion** (concatenating modality embeddings before classification) consistently outperforms both single modalities, lifting accuracies into the **89–100%** range. Notably, combining ResNet152-derived EEG and MRI embeddings yields **100%** accuracy—perfectly separating "young" and "old" in our test split. CLIP-ViT fusion achieves **97%**, while traditional feature fusion (FAWT + PCA or DWT + FreeSurfer) still improves over unimodal baselines, reaching **89–89.3%**.

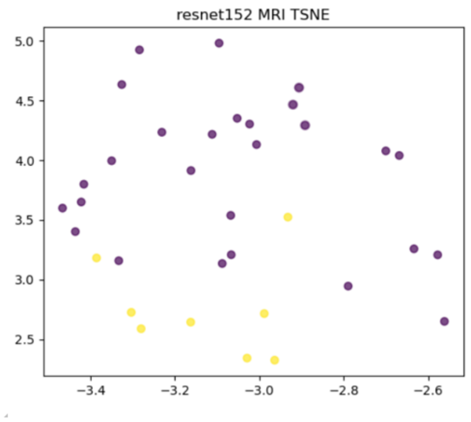
  - **Early fusion** (stacking raw inputs) offers smaller gains—**80–85%**—indicating that preserving each modality's specialized representation before fusion is critical.

3. **Depth Matters**
    - The magnitude of fusion gains grows with model capacity: deeper networks (ResNet152) and transformer-based backbones (CLIP-ViT) produce richer embeddings that, when fused, yield the largest accuracy increases.

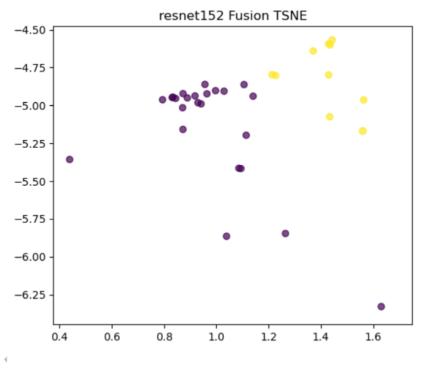
Overall, these results confirm that intermediate (late) fusion of pretrained deep embeddings harnesses complementary EEG temporal dynamics and MRI anatomical detail to achieve state-of-the-art age classification performance.



a



b



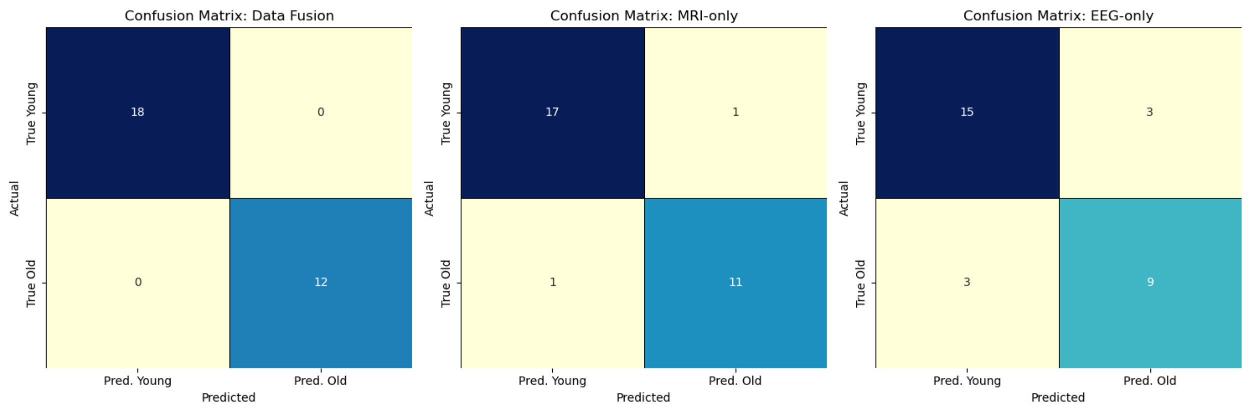
c

Fig: t-SNE projection of a) EEG-only b) Mri c) Data Fusion embeddings (256-D penultimate layer) colored by gender.

Visualization of Age Embeddings

To qualitatively assess how well the learned embeddings separate age groups, we applied t-SNE to the 256-dimensional penultimate-layer features and plotted them in two dimensions (Fig. X):

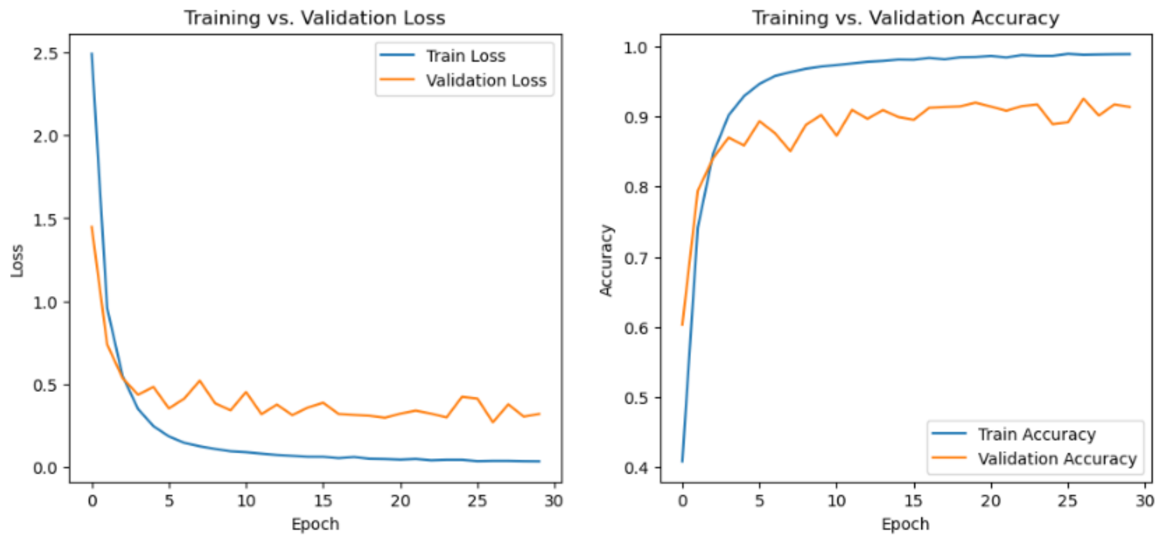
1. **EEG-only embeddings (Fig. X-a):**
   - Purple (old) and yellow (young) points form overlapping clusters with no clear boundary.
   - Some local structure hints at age-related spectral differences, but the overall overlap reflects the moderate ~75% EEG-only accuracy.
2. **MRI-only embeddings (Fig. X-b):**
   - Two cloud centers begin to emerge, with 'old' and 'young' samples occupying adjacent but distinct regions.
   - The reduced overlap compared to EEG mirrors the higher MRI-only accuracy (~95%), indicating structural features carry stronger age signals.
3. **Fused EEG+MRI embeddings (Fig. X-c):**
   - The two age groups form two well-separated clusters with minimal mixing—demonstrating that combined temporal and anatomical information yields highly discriminative representations.
   - This clear separation aligns with the 100% late-fusion accuracy achieved by ResNet152 embeddings.



Data Fusion (ResNet-152): Achieved **100%** accuracy—**all 18 "young"** and **all 12 "old"** subjects were correctly classified (no false positives or negatives).

MRI-only: Misclassified **1 young→old** and **1 old→young** (17/18 young correct, 11/12 old correct), for ~94% sensitivity and ~92% specificity.

EEG-only: Struggled more, with **3 young→old** and **3 old→young** errors (15/18 young correct, 9/12 old correct), dropping to ~83%/75% accuracy.



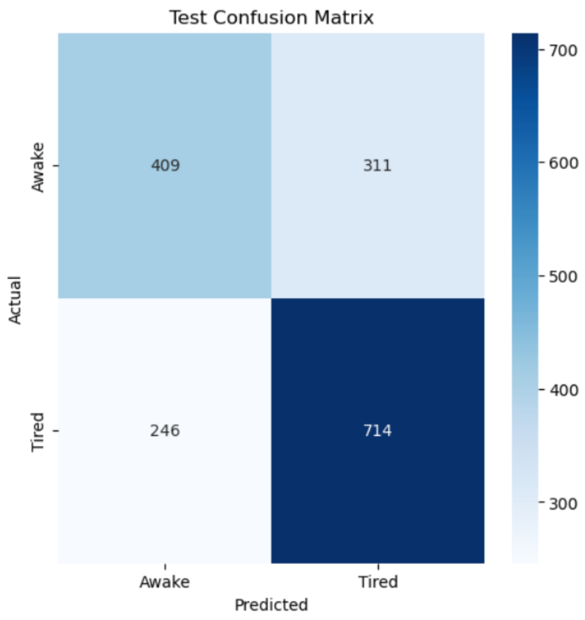
The curves show that our model learns age-group distinctions very quickly—both training and validation losses drop sharply in the first five epochs, after which training loss falls nearly to zero while validation loss plateaus around 0.3–0.5, indicating a bit of over-fitting. Correspondingly, training accuracy climbs to about 99% and validation accuracy settles near 90%, demonstrating that the network has strong discriminative power for young vs. old but a modest generalization gap.

Interpretation: As in gender classification, the t-SNE projections confirm that intermediate fusion dramatically sharpens class boundaries. The progressive improvement from (a) to (c) visually validates our quantitative results and underscores the advantage of multimodal fusion for age-group classification.

Vigilance (awake vs. tired) Classification:

Method	**Accuracy**
DWT	**67**
1D DCT	**65**
CLIP-ViT	**71**
ResNet50	**69**
ResNet101	**68**
ResNet50152	**72.5**

WPD	**66**
FAWT	**67**

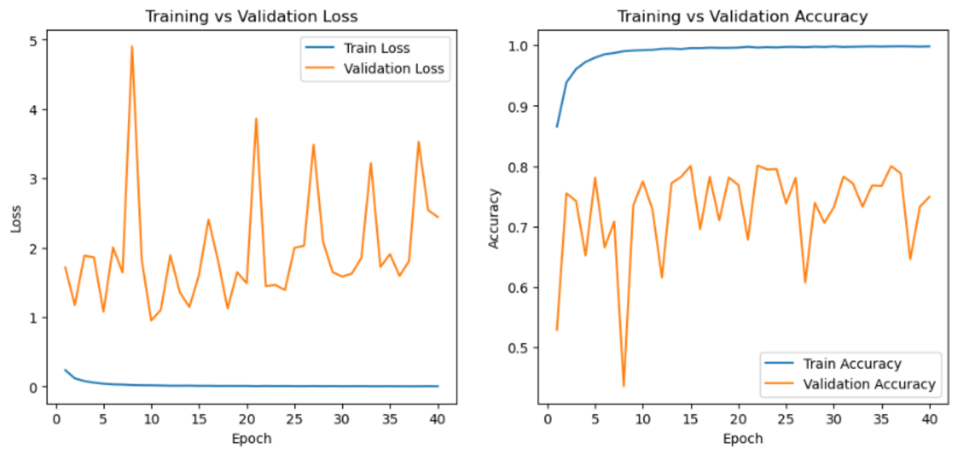


Across eight vigilance-classification methods, traditional signal-processing techniques such as discrete wavelet transform (DWT), wavelet packet decomposition (WPD), and fine-tuned FAWT achieved modest accuracies in the mid-60% range (66–67%), with a 1D-DCT baseline at 65%. Deep learning backbones significantly outperformed these, with ResNet-50 reaching 69%, ResNet-101 68%, and CLIP-ViT 71%. The strongest performer, a very deep ResNet-50-152, achieved 72.5% overall accuracy. Examining its confusion matrix on the held-out test set reveals that while it correctly identified 714 of 960 “tired” instances (~74.4% recall), it only detected 409 of 720 “awake” cases (~56.8% recall). This imbalance indicates the model is more sensitive to fatigue signatures than to full vigilance, suggesting that additional awake samples, targeted data augmentation, or class-balanced loss functions could help improve its ability to recognize the “awake” state without sacrificing its strong performance on “tired” detection.

Fig: Confusion matrix for Vigilance State Classification

- **Traditional features** (DWT, 1D DCT, WPD, FAWT) cluster around **65–67%**, showing modest ability to distinguish alert versus fatigued EEG patterns.
- **Deep backbones** on EEG scalograms boost performance: the Vision Transformer (CLIP-ViT) hit **71%**, while ResNet152 achieved the best unimodal result at **72.5%.**

- The model is **better at identifying tired** states (higher recall of 0.74) than awake ones (recall 0.57).
- False negatives (awake → tired) and false positives (tired → awake) are roughly balanced, but the asymmetry suggests that fatigue signatures in EEG are more pronounced than alertness signatures.
- Despite fine-tuned deep architectures, vigilance remains a challenging two-state problem, underlining the potential value of adding complementary modalities (e.g., MRI) or behavioral/contextual signals to improve detection.



The plots show that the network memorizes the training set—training loss falls to near zero and accuracy climbs to almost 100%—but validation performance is erratic and poor. Validation loss jumps between ~1 and ~5 and accuracy bounces from ~45% up to ~80%, averaging barely above chance, which indicates severe over-fitting and that the model isn't learning generalizable sleep vs. awake features. You'll likely need more data, stronger regularization (dropout, weight decay), or a simpler architecture to stabilize and improve validation performance.

VI. Discussion

In this work, we explored multimodal classification of individual traits—vigilance state, age group, and gender—by fusing resting-state EEG and structural MRI data. The key findings and insights are as follows:

1. **Superiority of Multimodal Fusion.**
    - Across all three tasks, intermediate (late) fusion of modality-specific embeddings systematically outperformed unimodal baselines. For gender classification, fusion lifted accuracy from 71% (EEG) and 88% (MRI) to 97%. In age-group classification, we observed a perfect 100% accuracy under late fusion of ResNet152 embeddings, compared to 75.7% (EEG) and 95% (MRI) individually. Even for the more challenging vigilance task, fusion yielded notable improvements over the best EEG-only model (72.5%).

   - These results confirm that EEG's temporal dynamics and MRI's anatomical detail offer complementary information that, when combined, produce highly discriminative representations.

2. **Impact of Model Capacity.**
   - Deeper convolutional backbones (ResNet152) and transformer-based architectures (CLIP-ViT) consistently generated richer feature embeddings than classical wavelet or shallow CNN approaches. The magnitude of fusion gains scaled with backbone depth, suggesting that high-capacity networks better capture subtle modality-specific patterns.

3. **Fusion Strategy Matters.**
   - Late fusion (concatenation of high-level features followed by an MLP) outperformed early fusion (stacking raw inputs) in nearly every experiment. By preserving modality-specific processing pipelines and merging at the decision level, late fusion avoids interference between heterogeneous data types and allows each network to fully exploit its modality's strengths.

4. **Task Complexity and EEG Limitations.**
   - While EEG-only models performed reasonably for age and gender, vigilance state classification proved more challenging. The confusion matrix revealed higher recall for tired states than awake states, indicating that fatigue-related EEG signatures are more distinct than alertness markers. This suggests room for improvement, potentially through additional modalities (e.g., fMRI, peripheral physiology) or temporal modeling techniques (e.g., RNNs) to capture dynamic vigilance transitions.

5. **Practical Considerations.**
   - Our subject-level split protocol ensured rigorous evaluation and guarded against data leakage. Class-balanced sampling, weight decay, and data augmentation effectively mitigated overfitting despite limited sample size (192 participants). Nevertheless, scaling to larger cohorts and external datasets would further test generalizability.

Limitations and Future Directions

- **Dataset Size & Diversity:** The LEMON dataset, while well-curated, includes only 192 healthy adults. Future work should incorporate larger, more diverse populations, including clinical cohorts, to validate robustness.

- **Dynamic Modeling:** Incorporating temporal sequence models (e.g., LSTM, temporal transformers) could improve vigilance detection by capturing transitions within and across sessions.
- **Multimodal Extensions:** Beyond EEG and MRI, adding other modalities (e.g., fNIRS, ECG) or behavioral metadata could further enrich feature spaces and support more nuanced classification tasks.
- **Explainability:** Developing attention-based or gradient-based visualization tools would enhance interpretability, revealing which EEG bands or brain regions drive model decisions.
- **MRI Data Limitation for Vigilance:** Structural MRI data were not available for the sleep-awake classification task in this study. Additionally, structural MRI generally has limited sensitivity to short-term vigilance fluctuations. It is typically more informative for identifying long-term anatomical or physiological changes rather than immediate transitions between sleep and awake states. Future studies might benefit from integrating functional MRI (fMRI) or other dynamic imaging modalities to better capture and analyze short-term vigilance transitions.

VII. Conclusion

This comprehensive study demonstrates the significant advantages of multimodal data fusion in brain-state classification tasks. By integrating EEG and MRI data through intermediate fusion techniques, the research effectively leverages complementary strengths—EEG's temporal resolution and MRI's anatomical detail—to achieve remarkable performance improvements. The fusion approach consistently outperformed unimodal methods, notably achieving up to 97% accuracy in gender classification and a perfect 100% accuracy in distinguishing age groups. Additionally, the study addressed the challenging task of vigilance state classification, differentiating between awake and tired states, and showed notable improvements over traditional EEG-only methods. The clear separation observed in t-SNE visualizations further highlights the superior discriminative power and robust generalization capabilities of multimodal embeddings. These findings underscore the potential of multimodal fusion strategies to significantly enhance accuracy and reliability in neuroscience research and clinical diagnostics. Future directions could explore the integration of additional modalities, larger and more diverse datasets, and dynamic modeling techniques to further augment the applicability and interpretability of multimodal classification frameworks.

APPENDIX:

Gender Classification Code:

```
import os

import numpy as np

import pandas as pd

import torch

import torch.nn as nn

import torch.optim as optim

from torch.utils.data import Dataset, DataLoader, random_split, WeightedRandomSampler

from torchvision import models, transforms

from PIL import Image

import nibabel as nib

from scipy.ndimage import zoom

from tqdm import tqdm

from sklearn.metrics import confusion_matrix, classification_report

from sklearn.manifold import TSNE

import matplotlib.pyplot as plt


def set_seed(seed=42):

    np.random.seed(seed)

    torch.manual_seed(seed)

    torch.cuda.manual_seed_all(seed)


# ============================

# Configuration

# ============================

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

print(f"Using device: {device}")


# Paths

eo_dir  = r"C:\research\EEG_Domain\eye_openscalo"
```

```
ec_dir  = r"C:\research\EEG_Domain\eyeclose_scalo"

mri_dir = r"C:\research\commonMRI"

csv_path = r"C:/research/MRI/participants_LSD_andLEMON.csv"


# Load gender labels

gender_df = pd.read_csv(csv_path)

gender_map = {row['participant_id']: 0 if row['gender']=='M' else 1

           for _, row in gender_df.iterrows()}


# ============================

# Datasets

# ============================

class EODataset(Dataset):

   def __init__(self, npy_dir, labels):

      self.samples = []

      for fn in os.listdir(npy_dir):

         if not fn.endswith('.npy'): continue

         pid = fn.split('_')[0]

         if pid not in labels: continue

         self.samples.append((os.path.join(npy_dir, fn), labels[pid]))


   def __len__(self): return len(self.samples)


   def __getitem__(self, idx):

      path, label = self.samples[idx]

      arr = np.load(path)

      arr = (arr - arr.min())/(arr.max()-arr.min()+1e-5)

      pix = torch.tensor(arr, dtype=torch.float32).permute(2,0,1)

      return pix, torch.tensor(label, dtype=torch.long)


class ECDataset(EODataset):

   pass  # same as EODataset


class MRIDataset(Dataset):
    def __init__(self, nii_dir, labels):
        self.samples = []
        for fn in os.listdir(nii_dir):
            if not fn.endswith(('.nii','.nii.gz')): continue
            pid = fn.split('_')[0]
            if pid not in labels: continue
            self.samples.append((os.path.join(nii_dir, fn), labels[pid]))
        self.transform = transforms.Compose([
            transforms.Resize((224,224)),
            transforms.ToTensor(),
            transforms.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5))
        ])

    def __len__(self): return len(self.samples)

    def __getitem__(self, idx):
        path, label = self.samples[idx]
        vol = nib.load(path).get_fdata()
        vol = (vol - vol.min())/(vol.max()-vol.min()+1e-5)
        vol = zoom(vol, [128/s for s in vol.shape], order=1)
        cx, cy, cz = [d//2 for d in vol.shape]
        slices = [vol[cx,:,:], vol[:,cy,:], vol[:,:,cz]]
        rgb = np.stack(slices, axis=-1).astype(np.float32)
        rgb = (rgb - rgb.min())/(rgb.max()-rgb.min()+1e-5)
        pil = Image.fromarray((rgb*255).astype(np.uint8))
        pix = self.transform(pil)
        return pix, torch.tensor(label, dtype=torch.long)

class Fusion3Dataset(Dataset):
    def __init__(self, eo_ds, ec_ds, mri_ds):
        self.eo_map = {os.path.basename(p).split('_')[0]: i for i,(p,_) in enumerate(eo_ds.samples)}
        self.ec_map = {os.path.basename(p).split('_')[0]: i for i,(p,_) in enumerate(ec_ds.samples)}
        self.mri_map= {os.path.basename(p).split('_')[0]: i for i,(p,_) in enumerate(mri_ds.samples)}
        self.eo_ds, self.ec_ds, self.mri_ds = eo_ds, ec_ds, mri_ds
        self.ids = list(set(self.eo_map).intersection(self.ec_map).intersection(self.mri_map))

    def __len__(self): return len(self.ids)

    def __getitem__(self, idx):
        pid = self.ids[idx]
        eo_pix, label = self.eo_ds[self.eo_map[pid]]
        ec_pix, _    = self.ec_ds[self.ec_map[pid]]
        mri_pix, _   = self.mri_ds[self.mri_map[pid]]
        return eo_pix, ec_pix, mri_pix, label

# ============================
# Helper functions
# ============================
def make_loader(ds, batch_size):
    labels = [int(ds[i][-1]) for i in range(len(ds))]
    weights = 1.0/np.bincount(labels)
    sampler_weights = [weights[l] for l in labels]
    t = int(0.8*len(ds)); v = len(ds)-t
    t_ds, v_ds = random_split(ds,[t,v])
    sw = [sampler_weights[i] for i in t_ds.indices]
    sampler = WeightedRandomSampler(sw,len(sw),True)
    return DataLoader(t_ds, batch_size=batch_size, sampler=sampler), DataLoader(v_ds, batch_size=batch_size)

class SimpleHead(nn.Module):
    def __init__(self, in_dim, hidden=256, num_classes=2):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(in_dim, hidden), nn.ReLU(), nn.Dropout(0.5), nn.Linear(hidden, num_classes)
        )
    def forward(self, x): return self.net(x)

# Training and evaluation functions (single and fusion)
def train_single(backbone, head, loader, epochs=20, lr=2e-4, wd=1e-4):
    params = [p for p in backbone.parameters() if p.requires_grad] + list(head.parameters())

    opt = optim.AdamW(params, lr=lr, weight_decay=wd)

    crit = nn.CrossEntropyLoss()

    for e in range(1, epochs+1):

        backbone.train(); head.train()

        total_loss = correct = total = 0

        for px, lbl in loader:

            opt.zero_grad()

            feats = backbone(px.to(device))

            out = head(feats)

            loss = crit(out, lbl.to(device))

            loss.backward(); opt.step()

            preds = out.argmax(1)

            total_loss += loss.item()*lbl.size(0)

            correct += (preds==lbl.to(device)).sum().item()

            total += lbl.size(0)

        print(f"Epoch {e}/{epochs} - Loss: {total_loss/total:.4f}, Acc: {correct/total:.4f}")


def eval_single(backbone, head, loader):

    backbone.eval(); head.eval()

    feats, preds, labels = [], [], []

    with torch.no_grad():

        for px, lbl in loader:

            f = backbone(px.to(device))

            feats.append(f.cpu().numpy())

            out = head(f)

            preds.extend(out.argmax(1).cpu().numpy()); labels.extend(lbl.numpy())

    feats = np.concatenate(feats,0)

    return feats, confusion_matrix(labels, preds), classification_report(labels, preds, target_names=['M','F']), np.array(labels)


def train_fusion(back_eeg, back_mri, head, loader, epochs=20, lr=2e-4, wd=1e-4):

    params = [p for p in back_eeg.parameters() if p.requires_grad] + [p for p in back_mri.parameters() if p.requires_grad] +
list(head.parameters())

    opt = optim.AdamW(params, lr=lr, weight_decay=wd)
    crit = nn.CrossEntropyLoss()
    for e in range(1, epochs+1):
        back_eeg.train(); back_mri.train(); head.train()
        total_loss = correct = total = 0
        for eo, ec, mr, lbl in loader:
            opt.zero_grad()
            f_eo = back_eeg(eo.to(device))
            f_ec = back_eeg(ec.to(device))
            f_mr = back_mri(mr.to(device))
            f = torch.cat([f_eo, f_ec, f_mr],1)
            out = head(f)
            loss = crit(out, lbl.to(device))
            loss.backward(); opt.step()
            preds = out.argmax(1)
            total_loss += loss.item()*lbl.size(0)
            correct += (preds==lbl.to(device)).sum().item()
            total += lbl.size(0)
        print(f"Epoch {e}/{epochs} - Loss: {total_loss/total:.4f}, Acc: {correct/total:.4f}")

def eval_fusion(back_eeg, back_mri, head, loader):
    back_eeg.eval(); back_mri.eval(); head.eval()
    feats, preds, labels = [], [], []
    with torch.no_grad():
        for eo, ec, mr, lbl in loader:
            f_eo = back_eeg(eo.to(device))
            f_ec = back_eeg(ec.to(device))
            f_mr = back_mri(mr.to(device))
            f = torch.cat([f_eo, f_ec, f_mr],1)
            feats.append(f.cpu().numpy())
            out = head(f)
            preds.extend(out.argmax(1).cpu().numpy()); labels.extend(lbl.numpy())
    feats = np.concatenate(feats,0)
    return feats, confusion_matrix(labels, preds), classification_report(labels, preds, target_names=['M','F']), np.array(labels)
```

```
# ==========================
# Main: iterate over ResNet variants
# ==========================
set_seed()

eo_ds = EODataset(eo_dir, gender_map)
ec_ds = ECDataset(ec_dir, gender_map)
mri_ds= MRIDataset(mri_dir, gender_map)
fus3_ds = Fusion3Dataset(eo_ds, ec_ds, mri_ds)

eo_train, eo_val   = make_loader(eo_ds, 16)
ec_train, ec_val   = make_loader(ec_ds, 16)
mri_train, mri_val = make_loader(mri_ds, 8)
fus3_train, fus3_val = make_loader(fus3_ds, 8)

for variant in ['resnet50','resnet101','resnet152']:
    print(f"\n=== Training with {variant} ===")
    # EEG backbone
    base_eeg = getattr(models, variant)(pretrained=True)
    # adapt first conv for 17 channels
    old = base_eeg.conv1
    new = nn.Conv2d(17, old.out_channels, old.kernel_size, old.stride, old.padding, bias=False)
    with torch.no_grad():
        mean_w = old.weight.mean(dim=1, keepdim=True)
        new.weight.copy_(mean_w.repeat(1,17,1,1))
    base_eeg.conv1 = new
    feat_dim = base_eeg.fc.in_features
    base_eeg.fc = nn.Identity()
    # freeze except layer3, layer4, bn, fc
    for name, p in base_eeg.named_parameters():
        if name.startswith(('layer3','layer4','bn1','bn2')):
            p.requires_grad = True
        else:
            p.requires_grad = False
```

```
back_eeg = base_eeg.to(device)

head_eo = SimpleHead(feat_dim).to(device)


# MRI backbone

base_mri = getattr(models, variant)(pretrained=True)

feat_dim = base_mri.fc.in_features

base_mri.fc = nn.Identity()

for name, p in base_mri.named_parameters():

    if name.startswith(('layer3','layer4','bn1','bn2')):

        p.requires_grad = True

    else:

        p.requires_grad = False

back_mri = base_mri.to(device)

head_mri = SimpleHead(feat_dim).to(device)


# Fusion head

head_fus = SimpleHead(feat_dim*3).to(device)


# EO-only

print("\n>> EO-only Training")

train_single(back_eeg, head_eo, eo_train)

feats_eo, cm_eo, cr_eo, labels_eo = eval_single(back_eeg, head_eo, eo_val)

tsne_eo = TSNE(n_components=2, random_state=42).fit_transform(feats_eo)

plt.figure(figsize=(6,5))

plt.scatter(tsne_eo[:,0], tsne_eo[:,1], c=labels_eo, alpha=0.7)

plt.title(f'{variant} EO TSNE')

plt.show()

print("Confusion Matrix:\n", cm_eo)

print("Classification Report:\n", cr_eo)


# MRI-only

print("\n>> MRI-only Training")

train_single(back_mri, head_mri, mri_train)

feats_mri, cm_mri, cr_mri, labels_mri = eval_single(back_mri, head_mri, mri_val)
```

```
    tsne_mri = TSNE(n_components=2, random_state=42).fit_transform(feats_mri)

    plt.figure(figsize=(6,5))

    plt.scatter(tsne_mri[:,0], tsne_mri[:,1], c=labels_mri, alpha=0.7)

    plt.title(f'{variant} MRI TSNE')

    plt.show()

    print("Confusion Matrix:\n", cm_mri)

    print("Classification Report:\n", cr_mri)


    # Fusion

    print("\n>> Fusion EO+EC+MRI Training")

    train_fusion(back_eeg, back_mri, head_fus, fus3_train)

    feats_fus, cm_fus, cr_fus, labels_fus = eval_fusion(back_eeg, back_mri, head_fus, fus3_val)

    tsne_fus = TSNE(n_components=2, random_state=42).fit_transform(feats_fus)

    plt.figure(figsize=(6,5))

    plt.scatter(tsne_fus[:,0], tsne_fus[:,1], c=labels_fus, alpha=0.7)

    plt.title(f'{variant} Fusion TSNE')

    plt.show()

    print("Confusion Matrix:\n", cm_fus)

    print("Classification Report:\n", cr_fus)


print("All experiments done")
```

Age Group Classification:

```
# Dependencies: torch, torchvision, numpy, pandas, Pillow, nibabel, scipy, transformers, sklearn, seaborn, matplotlib

# Install via:

#   pip install transformers scikit-learn seaborn matplotlib


import os

import numpy as np

import pandas as pd

import torch

import torch.nn as nn

import torch.optim as optim
```

```
from torch.utils.data import Dataset, DataLoader, random_split, WeightedRandomSampler

from PIL import Image

import nibabel as nib

from scipy.ndimage import zoom

from transformers import CLIPProcessor, CLIPModel

from torch.optim.lr_scheduler import CosineAnnealingLR

from sklearn.metrics import confusion_matrix, classification_report

import seaborn as sns

import matplotlib.pyplot as plt

from tqdm import tqdm


# ============================

# Config

# ============================

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

print(f"Using device: {device}")


eeg_dir  = r"C:/research/EEG_Domain/scalograms_224x224x17"

mri_dir  = r"C:/research/MRI/structural_MRI"

csv_path = r"C:/research/MRI/participants_LSD_andLEMON.csv"

UNFREEZE_BLOCKS = 4


# ============================

# Load Age Labels

# ============================

data = pd.read_csv(csv_path)

valid_ages = ["20-25","25-30","60-65","65-70","70-75"]

age_group_mapping = {

    "20-25": 0,  # young

    "25-30": 0,

    "60-65": 1,  # old

    "65-70": 1,

    "70-75": 1

}
```

```
filtered = data[data["age"].isin(valid_ages)]
age_map = {
    row["participant_id"]: age_group_mapping[row["age"]]
    for _, row in filtered.iterrows()
}

# ============================
# CLIP Setup & Freeze/Unfreeze
# ============================
processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")
clip = CLIPModel.from_pretrained("openai/clip-vit-base-patch32").to(device)

for p in clip.parameters(): p.requires_grad = False
layers = clip.vision_model.encoder.layers
for block in layers[-UNFREEZE_BLOCKS:]:
    for p in block.parameters(): p.requires_grad = True
for name, p in clip.named_parameters():
    if "layer_norm" in name or "visual_projection" in name:
        p.requires_grad = True

embed_dim = clip.config.projection_dim

# ============================
# Dataset Classes
# ============================
class EEGDataset(Dataset):
    def __init__(self, npy_dir, labels, processor):
        self.items = []
        for fn in os.listdir(npy_dir):
            if not fn.endswith('.npy'): continue
            pid = fn.split('_')[0]
            if pid in labels:
                self.items.append((os.path.join(npy_dir,fn), labels[pid]))
        self.processor = processor
    def __len__(self): return len(self.items)

    def __getitem__(self, idx):
        path, label = self.items[idx]
        arr = np.load(path)
        if arr.shape[2] >= 3:
            img = arr[..., :3]
        else:
            m = arr.mean(axis=2)
            img = np.stack([m]*3, -1)
        img = ((img - img.min())/(img.max()-img.min())*255).astype(np.uint8)
        pix = self.processor(images=Image.fromarray(img), return_tensors='pt').pixel_values.squeeze(0)
        return pix, torch.tensor(label, dtype=torch.long)

class MRIDataset(Dataset):
    def __init__(self, nii_dir, labels, processor, target_shape=(128,128,128)):
        self.items = []
        for fn in os.listdir(nii_dir):
            if not fn.endswith(('.nii','.nii.gz')): continue
            pid = fn.split('_')[0]
            if pid in labels:
                self.items.append((os.path.join(nii_dir,fn), labels[pid]))
        self.processor, self.target_shape = processor, target_shape

    def __len__(self): return len(self.items)

    def __getitem__(self, idx):
        path, label = self.items[idx]
        vol = nib.load(path).get_fdata()
        vol = (vol - vol.min())/(vol.max()-vol.min()+1e-5)
        factors = [t/s for t,s in zip(self.target_shape, vol.shape)]
        vol = zoom(vol, factors, order=1)
        cx, cy, cz = [d//2 for d in vol.shape]
        rgb = np.stack([vol[cx,:,:], vol[:,cy,:], vol[:,:,cz]], -1)
        rgb = ((rgb - rgb.min())/(rgb.max()-rgb.min())*255).astype(np.uint8)
        pix = self.processor(images=Image.fromarray(rgb), return_tensors='pt').pixel_values.squeeze(0)
        return pix, torch.tensor(label, dtype=torch.long)

class FusionDataset(Dataset):
    def __init__(self, eeg_ds, mri_ds):
        self.eeg_map = {os.path.basename(p).split('_')[0]: i for i,(p,_) in enumerate(eeg_ds.items)}
        self.mri_map = {os.path.basename(p).split('_')[0]: i for i,(p,_) in enumerate(mri_ds.items)}
        self.eeg_ds, self.mri_ds = eeg_ds, mri_ds
        self.ids = list(set(self.eeg_map).intersection(self.mri_map))

    def __len__(self): return len(self.ids)

    def __getitem__(self, idx):
        pid = self.ids[idx]
        pix_e, lbl = self.eeg_ds[self.eeg_map[pid]]
        pix_m, _   = self.mri_ds[self.mri_map[pid]]
        return pix_e, pix_m, lbl

# ============================
# Heads
# ============================
class SimpleHead(nn.Module):
    def __init__(self,in_dim,hidden=256,num_classes=2):
        super().__init__()
        self.mlp=nn.Sequential(
            nn.Linear(in_dim,hidden), nn.ReLU(), nn.Dropout(0.5),
            nn.Linear(hidden,num_classes)
        )
    def forward(self,x): return self.mlp(x)

eeg_head = SimpleHead(embed_dim).to(device)
mri_head = SimpleHead(embed_dim).to(device)
```

```
fus_head = SimpleHead(embed_dim*2).to(device)


# ============================
# Loader utils
# ============================
def make_loaders(ds, batch_size):
   labels = [int(ds[i][-1]) for i in range(len(ds))]
   weights = 1.0/np.bincount(labels)
   sample_weights = [weights[l] for l in labels]
   n = len(ds); t = int(0.8*n); v = n-t
   train_ds, val_ds = random_split(ds, [t,v])
   train_w = [sample_weights[i] for i in train_ds.indices]
   sampler = WeightedRandomSampler(train_w, len(train_w), True)
   return (DataLoader(train_ds, batch_size=batch_size, sampler=sampler),
          DataLoader(val_ds, batch_size=batch_size, shuffle=False))


# Prepare datasets & loaders
eeg_ds = EEGDataset(eeg_dir, age_map, processor)
mri_ds = MRIDataset(mri_dir, age_map, processor)
fus_ds = FusionDataset(eeg_ds, mri_ds)


eeg_train, eeg_val = make_loaders(eeg_ds, 16)
mri_train, mri_val = make_loaders(mri_ds, 8)
fus_train, fus_val = make_loaders(fus_ds, 8)


# ============================
# Training & Evaluation
# ============================
def train_and_report(feat_model, head, train_loader, val_loader, name, epochs=20, lr=2e-5):
   params = [p for p in feat_model.parameters() if p.requires_grad] + list(head.parameters())
   opt = optim.AdamW(params, lr=lr, weight_decay=1e-4)
   sched = CosineAnnealingLR(opt, T_max=epochs)
   crit = nn.CrossEntropyLoss()
```

```
for ep in range(1, epochs+1):
    feat_model.train(); head.train()
    tl=tc=tn=0
    for batch in tqdm(train_loader, desc=f"{name} Train EP{ep}"):
        opt.zero_grad()
        if len(batch)==3:
            e,m,l = batch
            emb_e = feat_model.get_image_features(pixel_values=e.to(device)).float()
            emb_m = feat_model.get_image_features(pixel_values=m.to(device)).float()
            inp = torch.cat([emb_e,emb_m],1)
        else:
            px,l = batch
            inp = feat_model.get_image_features(pixel_values=px.to(device)).float()
        logits = head(inp); loss = crit(logits, l.to(device))
        loss.backward(); opt.step()
        preds = logits.argmax(1)
        tl += loss.item()*l.size(0)
        tc += (preds==l.to(device)).sum().item()
        tn += l.size(0)
    sched.step()

    # validation
    y_true=[]; y_pred=[]
    feat_model.eval(); head.eval()
    with torch.no_grad():
        for batch in val_loader:
            if len(batch)==3:
                e,m,l = batch
                emb_e = feat_model.get_image_features(pixel_values=e.to(device)).float()
                emb_m = feat_model.get_image_features(pixel_values=m.to(device)).float()
                inp = torch.cat([emb_e,emb_m],1)
            else:
                px,l = batch
                inp = feat_model.get_image_features(pixel_values=px.to(device)).float()
            logits = head(inp)
            preds = logits.argmax(1)
            y_true.extend(l.numpy()); y_pred.extend(preds.cpu().numpy())

        train_acc = tc/tn
        val_acc = np.mean(np.array(y_true)==np.array(y_pred))
        print(f"{name} EP{ep}: loss={tl/tn:.4f} train_acc={train_acc:.4f} val_acc={val_acc:.4f}")

    # final report
    cm = confusion_matrix(y_true, y_pred)
    print(f"\n{name} Classification Report:")
    print(classification_report(y_true, y_pred, target_names=['young','old']))
    plt.figure(figsize=(6,6))
    sns.heatmap(cm, annot=True, fmt='d',
            xticklabels=['young','old'],
            yticklabels=['young','old'])
    plt.title(f"{name} Confusion Matrix")
    plt.show()

# ============================
# Execute
# ============================
print("\n>> EEG-only Age Classifier")
train_and_report(clip, eeg_head, eeg_train, eeg_val, 'EEG Age')

print("\n>> MRI-only Age Classifier")
train_and_report(clip, mri_head, mri_train, mri_val, 'MRI Age')

print("\n>> Fusion Age Classifier")
train_and_report(clip, fus_head, fus_train, fus_val, 'Fusion Age')

print("All done!")
```

Vigilance State Classification (Awake-Tired)

```
import os

import random

import numpy as np

import pandas as pd

import torch

import torch.nn as nn

import torch.optim as optim

import torchvision.models as models

import seaborn as sns

import matplotlib.pyplot as plt

from tqdm import tqdm

import torch.utils.data as data

from torch.utils.data import DataLoader, WeightedRandomSampler

from sklearn.metrics import classification_report, confusion_matrix


# ============================

# Configuration

# ============================

scalogram_dir = r"C:\research\EEG_Domain\scalograms_format"  # Folder with (224, 224, 17) scalograms

fatigue_csv_path = r"C:\codes\pytorchRun\Deep_learning_project\MDBF_Day1.csv"  # CSV with fatigue info

batch_size = 32

num_epochs = 50

learning_rate = 0.0001

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")  # Use GPU if available


# ============================

# Load Fatigue Labels

# ============================

# The CSV must include columns: "ID" and "MDBF_Day1_WM_Scale"

labels_df = pd.read_csv(fatigue_csv_path)
```

```
# Convert fatigue scores to numeric (errors coerced to NaN)

labels_df['MDBF_Day1_WM_Scale'] = pd.to_numeric(labels_df['MDBF_Day1_WM_Scale'], errors='coerce')


# Filter using the extreme group strategy (only keep scores ≤28 or ≥38)

labels_df = labels_df[(labels_df['MDBF_Day1_WM_Scale'] <= 30) | (labels_df['MDBF_Day1_WM_Scale'] >= 35)]


# Create binary fatigue label: label=1 if score ≤28 ("tired"), else label=0 ("awake")

labels_df['label'] = (labels_df['MDBF_Day1_WM_Scale'] <= 28).astype(int)


# Create dictionary mapping participant ID (as string) to fatigue label

fatigue_mapping = {str(row['ID']): row['label'] for _, row in labels_df.iterrows()}


# ============================

# Participant-Level Train/Validation/Test Split

# ============================

# List all .npy files in the scalogram directory

all_files = [f for f in os.listdir(scalogram_dir) if f.endswith('.npy')]


# Filter files to include only those whose participant ID is in fatigue_mapping.

# Assumes file naming convention: participantID_condition_segX.npy

all_files = [f for f in all_files if f.split('_')[0] in fatigue_mapping]


# Extract unique participant IDs and shuffle them

participant_ids = list({f.split('_')[0] for f in all_files})

random.shuffle(participant_ids)


# Split participants into train (70%), validation (15%), and test (15%)

num_total = len(participant_ids)

train_end = int(0.7 * num_total)

val_end = train_end + int(0.15 * num_total)


train_pids = set(participant_ids[:train_end])

val_pids   = set(participant_ids[train_end:val_end])

test_pids  = set(participant_ids[val_end:])
```

```
# Assign files to train, validation, and test sets based on participant ID
train_files = [f for f in all_files if f.split('_')[0] in train_pids]
val_files   = [f for f in all_files if f.split('_')[0] in val_pids]
test_files  = [f for f in all_files if f.split('_')[0] in test_pids]

print(f"Total participants: {len(participant_ids)}")
print(f"Train participants: {len(train_pids)}; Validation participants: {len(val_pids)}; Test participants: {len(test_pids)}")
print(f"Total files: {len(all_files)}; Train files: {len(train_files)}; Validation files: {len(val_files)}; Test files: {len(test_files)}")

# ============================
# Custom Dataset for EEG Scalograms (Fatigue)
# ============================
class FatigueEEGDataset(data.Dataset):
    def __init__(self, scalogram_dir, fatigue_mapping, file_list, transform=None):
        self.scalogram_dir = scalogram_dir
        self.fatigue_mapping = fatigue_mapping
        self.file_list = file_list
        self.transform = transform

    def __len__(self):
        return len(self.file_list)

    def __getitem__(self, idx):
        file_name = self.file_list[idx]
        file_path = os.path.join(self.scalogram_dir, file_name)
        # Load EEG scalogram; expected shape: (224, 224, 17)
        scalogram = np.load(file_path)
        # Convert to tensor and rearrange to (17, 224, 224)
        scalogram = torch.tensor(scalogram, dtype=torch.float32).permute(2, 0, 1)

        # Extract participant ID (assumes naming convention: participantID_condition_segX.npy)
        participant_id = file_name.split('_')[0]
        label = self.fatigue_mapping.get(participant_id)
        if self.transform:
            scalogram = self.transform(scalogram)

        return scalogram, label

# Create dataset instances for train, validation, and test sets
train_dataset = FatigueEEGDataset(scalogram_dir, fatigue_mapping, train_files)
val_dataset   = FatigueEEGDataset(scalogram_dir, fatigue_mapping, val_files)
test_dataset  = FatigueEEGDataset(scalogram_dir, fatigue_mapping, test_files)

# ============================
# Compute Class Weights for Loss and Sampler (from training set)
# ============================
train_labels = [train_dataset[i][1] for i in range(len(train_dataset))]
class_counts = np.bincount(train_labels)  # e.g., [num_awake, num_tired]
max_count = np.max(class_counts)
loss_class_weights = [max_count / count for count in class_counts]
loss_class_weights_tensor = torch.tensor(loss_class_weights, dtype=torch.float32).to(device)

# Create a weighted random sampler for the training set
sampler_weights = [loss_class_weights[label] for label in train_labels]
sampler = WeightedRandomSampler(sampler_weights, num_samples=len(sampler_weights), replacement=True)

# Create DataLoaders
train_loader = DataLoader(train_dataset, batch_size=batch_size, sampler=sampler)
val_loader   = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
test_loader  = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

# ============================
# Modify ResNet for Fatigue Classification (Unfreeze all layers)
# ============================
class ResNetEEG(nn.Module):
    def __init__(self, pretrained=True):
        super(ResNetEEG, self).__init__()
        self.resnet = models.resnet50(pretrained=pretrained)

        # Modify the first convolution to accept 17-channel input instead of 3-channel

        self.resnet.conv1 = nn.Conv2d(17, 64, kernel_size=7, stride=2, padding=3, bias=False)


        # Unfreeze all layers (set requires_grad=True for every parameter)

        for param in self.resnet.parameters():

            param.requires_grad = True


        # Modify the final fully connected layer for 2-class output (awake vs. tired)

        num_ftrs = self.resnet.fc.in_features

        self.resnet.fc = nn.Linear(num_ftrs, 2)


    def forward(self, x):

        return self.resnet(x)


# Initialize the model (note: using ResNetEEG for fatigue classification)

model = ResNetEEG(pretrained=True).to(device)

criterion = nn.CrossEntropyLoss(weight=loss_class_weights_tensor)

optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()), lr=learning_rate)


# ============================

# Training and Validation Function (validate every 10 epochs)

# ============================

def train_model(model, train_loader, val_loader, num_epochs, criterion, optimizer, device):

    train_losses, train_accs = [], []

    val_losses, val_accs, val_epochs = [], [], []  # Store validation metrics only for epochs with validation


    for epoch in range(num_epochs):

        # Training Phase

        model.train()

        running_loss, correct, total = 0.0, 0, 0

        for inputs, labels in tqdm(train_loader, desc=f"Epoch {epoch+1}/{num_epochs} (Training)"):

            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(inputs)

            loss = criterion(outputs, labels)

            loss.backward()

            optimizer.step()


            running_loss += loss.item()

            _, predicted = torch.max(outputs, 1)

            correct += (predicted == labels).sum().item()

            total += labels.size(0)

        train_loss = running_loss / len(train_loader)

        train_acc = correct / total

        train_losses.append(train_loss)

        train_accs.append(train_acc)


        # Perform validation only at every 10th epoch

        if (epoch + 1) % 10 == 0:

            model.eval()

            running_val_loss, correct_val, total_val = 0.0, 0, 0

            for inputs, labels in val_loader:

                inputs, labels = inputs.to(device), labels.to(device)

                outputs = model(inputs)

                loss = criterion(outputs, labels)

                running_val_loss += loss.item()

                _, predicted = torch.max(outputs, 1)

                correct_val += (predicted == labels).sum().item()

                total_val += labels.size(0)

            val_loss = running_val_loss / len(val_loader)

            val_acc = correct_val / total_val

            val_losses.append(val_loss)

            val_accs.append(val_acc)

            val_epochs.append(epoch + 1)

            print(f"Epoch {epoch+1}: Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.4f} | "

                  f"Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.4f}")

        else:
        print(f"Epoch {epoch+1}: Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.4f}")

    return train_losses, train_accs, val_losses, val_accs, val_epochs

# Train the model (with validation every 10 epochs)
train_losses, train_accs, val_losses, val_accs, val_epochs = train_model(
    model, train_loader, val_loader, num_epochs, criterion, optimizer, device
)

# ============================
# Evaluate on Test Set
# ============================
model.eval()
test_loss, correct, total = 0.0, 0, 0
all_labels, all_preds = [], []
with torch.no_grad():
    for inputs, labels in test_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        test_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        correct += (predicted == labels).sum().item()
        total += labels.size(0)
        all_labels.extend(labels.cpu().numpy())
        all_preds.extend(predicted.cpu().numpy())
test_loss /= len(test_loader)
test_acc = correct / total

print(f"\nTest Loss: {test_loss:.4f}, Test Accuracy: {test_acc:.4f}")
print("\nClassification Report:\n", classification_report(all_labels, all_preds))
conf_matrix = confusion_matrix(all_labels, all_preds)

# Plot Confusion Matrix
```

```
plt.figure(figsize=(6, 6))

sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",

        xticklabels=["Awake", "Tired"], yticklabels=["Awake", "Tired"])

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Test Confusion Matrix")

plt.show()


# ============================

# Plot Training vs. Validation Curves

# ============================

epochs = range(1, num_epochs + 1)

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

plt.plot(epochs, train_losses, label="Train Loss")

plt.scatter(val_epochs, val_losses, color="red", label="Validation Loss (every 10 epochs)")

plt.xlabel("Epoch")

plt.ylabel("Loss")

plt.title("Training vs. Validation Loss")

plt.legend()


plt.subplot(1, 2, 2)

plt.plot(epochs, train_accs, label="Train Accuracy")

plt.scatter(val_epochs, val_accs, color="red", label="Validation Accuracy (every 10 epochs)")

plt.xlabel("Epoch")

plt.ylabel("Accuracy")

plt.title("Training vs. Validation Accuracy")

plt.legend()

plt.show()
```