**1. Introduction**

In physiological signal analysis, **autoencoders** are powerful neural networks that learn a low-dimensional, compressed representation (embedding) of high-dimensional data, while aiming to reconstruct the original input as closely as possible. This ability to learn “meaningful” compressed features can be particularly useful when you have multiple sources of physiological data (e.g., chest device signals and wrist device signals) and need to fuse them or reduce dimensionality for downstream tasks (such as stress detection).

**1.1 Why Use Autoencoders for Physiological Signals?**

* **Noise Reduction**: Physiological signals (ECG, EMG, EDA, etc.) often contain noise. Autoencoders learn compressed latent representations that naturally discard some noise.
* **Feature Extraction**: The compressed representation (latent space) can be used as learned features for classification or regression tasks.
* **Multimodal Fusion**: Autoencoders can help unify feature extraction across multiple sensors (e.g., chest-worn vs. wrist-worn devices).

**2. Autoencoder Background**

A vanilla autoencoder consists of two primary components:

1. Encoder: Learns a mapping fθ(x) that compresses the input x into a lower-dimensional hidden representation z.

2. Decoder: Learns a mapping gϕ(z) that reconstructs x from the compressed representation z.

Formally, if x is an input vector (or multidimensional array in the case of signals), the encoder produces:

z = fθ(x)

where z ∈ ℝ^d is the latent (hidden) representation, typically with d < dim(x).

The decoder tries to recover x from z:

x̂ = gϕ(z) = gϕ(fθ(x))

The objective is to minimize the reconstruction error:

L(x, x̂) = ||x - x̂||²

or some variant (like mean squared error, mean absolute error, etc.). In Keras/TensorFlow, we typically specify loss="mse" for autoencoders.

**2.1 Convolutional Autoencoder**

In a **Convolutional Autoencoder (CAE)**, the encoder and decoder are primarily built from convolutional layers (and possibly pooling or upsampling layers). For 1D physiological signals:

* **Encoder**: Uses Conv1D layers (optionally with batch normalization) to learn spatial/temporal patterns.
* **Flatten + Dense**: Often the final convolution output is flattened, then projected into a smaller latent dimension.
* **Decoder**: Mirrors the structure of the encoder, replacing Conv1D (with stride or pooling) by transposed convolutions or upsampling. In the simplest case (as in the code below), we reshape a Dense output back to the convolutional shape and apply Conv1D.

Diagrammatically:

pgsql

CopyEdit

+----------------+

| |

| Input Signal | (e.g., shape = (5, 175) -> 5 features, 175 time points)

| |

+----------------+

|

v

+----------------+ <--- Convolutional + Activation

| Conv1D Layer |

| + BatchNorm |

+----------------+

|

v

+----------------+ <--- Convolutional + Activation

| Conv1D Layer |

| + BatchNorm |

+----------------+

|

v

+----------------+ <--- Flatten

| Flatten |

+----------------+

|

v

+----------------+ <--- Dense -> Latent Vector

| Dense Layer | (Latent space, e.g. 80 units)

+----------------+

|

| (Encoded Representation z)

|

v

+----------------+ <--- Dense + Reshape

| Dense Layer | (expanding back to original shape)

+----------------+

|

v

+----------------+ <--- Conv1D + Activation

| Conv1D Layer |

| + BatchNorm |

+----------------+

|

v

+----------------+ <--- Conv1D + Sigmoid -> Reconstructed Signal

| Conv1D Layer |

+----------------+

|

v

+----------------+

|Reconstruction |

+----------------+

**3. System Workflow**

We are dealing with three sets of signals:

1. **Chest** device (ECG, EMG, EDA, TEMP, RESP) — sampled at 700 Hz.
2. **Wrist** device BVP — sampled at 64 Hz.
3. **Wrist** device EDA and TEMP — sampled at 4 Hz.

The approach:

1. **Leave-One-Subject-Out (LOSO)** training for the chest autoencoder:
   * For each subject, train the autoencoder on all other subjects’ data (chest signals), then save the encoder.
   * The reconstruction loss helps the autoencoder learn a good representation of chest signals.
2. **Extract Features**:
   * For each subject (as the test subject):
     + Load the trained chest autoencoder (encoder).
     + Train two additional autoencoders (on that subject’s training partition) for BVP and EDA+TEMP, respectively.
     + Generate embeddings from chest, BVP, and EDA+TEMP signals.
     + Concatenate these embeddings with the label (or label distribution) for each window.
     + Save them for downstream tasks.

**3.1 Data Preparation and Windowing**

* Each subject’s signals are segmented into windows of **batch\_size = (sampling\_frequency × window\_seconds)**.
* For chest: batch\_size\_chest = 700 \* 0.25 = 175.
* For BVP: batch\_size\_bvp = 64 \* 0.25 = 16.
* For EDA/TEMP: batch\_size\_eda = 4 \* 0.25 = 1.

**Note**: If you feed these windowed segments as the “batch size” in Keras, you may observe slower training (because Keras sees a huge number of “steps per epoch”). One may want to separate the concept of “window size” from the “minibatch size” used in training (e.g., 32, 64). But for brevity, we keep it as is in the code.

**4. Key Formulas**

1. 1D Convolution:  
 Conv1D(x)[i, k] = σ(bₖ + Σ x[i+j] · w\_{j,k})

2. Batch Normalization:  
 x̂ = (x - μ) / sqrt(σ² + ε) · γ + β

3. Flatten & Dense:  
 Flatten(X) = vec(X)  
 Dense(h) = σ(Wh + b)

4. Loss:  
 L(x, x̂) = ||x - x̂||²

**5. Code Explanation**

Below is a streamlined version of the code with added comments and docstrings to clarify its functioning.

**5.1 Code Highlights**

1. **build\_autoencoder\_chest**
   * Creates an encoder of two Conv1D layers, each with window\_size filters, followed by flatten + dense.
   * The decoder uses a Dense + reshape + two Conv1D layers to reconstruct the input.
   * *Important details*:
     + batch\_size from the data is used as the number of filters in each Conv1D. This can be large. Consider using a smaller filter count for faster training.
2. **build\_autoencoder\_bvp** and **build\_autoencoder\_eda\_temp**
   * Similar concept, but different latent dimensionalities (like 40 for BVP, 4 for EDA/TEMP).
3. **train\_chest\_autoencoders**
   * For each subject, we do a leave-one-subject-out. The subject is withheld from training data, and we train on all the other subjects.
   * We then save the trained encoder for chest signals.
4. **extract\_features**
   * For each subject, we:
     + Load the chest encoder trained from LOSO.
     + Build separate autoencoders for BVP and EDA+TEMP and train them on the training portion of the wrist data.
     + Generate embeddings from chest, BVP, and EDA+TEMP.
     + Concatenate embeddings with the subject’s label for each window and save.

**6. Practical Considerations**

* **Minibatch vs. Window Size**:  
  In the current code, the “window size” (number of time samples per segment) is used as the number of filters in the convolution. Additionally, Keras sees each time-segment as a separate “batch item.” This can slow training if the dataset is large.  
  A more standard approach is to fix the convolution filters to a smaller, more typical number (like 32 or 64) and keep the window size as the time dimension only.
* **Mixed Precision**:  
  On Apple Silicon, enabling mixed precision can drastically speed training.

python

CopyEdit

from tensorflow.keras.mixed\_precision import experimental as mixed\_precision

policy = mixed\_precision.Policy('mixed\_float16')

mixed\_precision.set\_policy(policy)

* **Hyperparameters**:
  + **Kernel sizes**: (3, 6, 4, etc.) can be tuned.
  + **Latent dimension**: 80 for chest, 40 for BVP, 4 for EDA/TEMP may be adjusted based on dataset size or performance needs.
  + **Learning rate**: 0.00025 might be changed to 0.001 or 0.0001 to see if it converges faster/better.
* **Data Labeling**:  
  The code does a majority vote to determine the label for each window. This is a common approach, but you may consider alternative strategies.

**7. Summary**

This document provided:

1. **Theoretical Overview** of (Convolutional) Autoencoders
2. **Architectural Diagram** for a typical 1D convolutional autoencoder
3. **Key Mathematical Formulas**, including convolution, batch normalization, and autoencoder losses
4. **Annotated Code** that implements a leave-one-subject-out training scheme for chest signals and trains additional autoencoders for BVP and EDA/TEMP data, ultimately creating fused embeddings for each subject window.

Convolutional autoencoders are especially useful for **noise reduction**, **dimensionality reduction**, and **feature extraction** in physiological signals, enabling robust multimodal analysis, especially for tasks like stress detection or biometric authentication.