EEG-to-Robot Arm:
Real-Time Signal-Processing Pipeline

Joshua Chua

July 2025

Contents

1 High-Level Pipeline

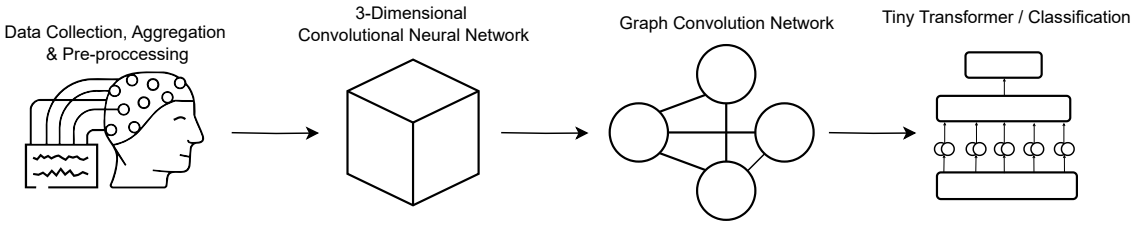

Figure 1: High level representation of the pipeline.

Figure 1 sketches the high-level processing stages discussed in this guide:

1. **Raw EEG Acquisition**
2. **Cleaning & Windowing**
3. **Spectrogram Cube Creation**
4. **3-D Depthwise CNN**
5. **Graph Convolutional Network (GCN)**
6. **Temporal Model: TCN or Tiny Transformer**
7. **Prediction Head**
8. **Command Decoding**
9. **Robot-Arm Control Loop**
10. **Feedback & Adaptation**

2 EEG Acquisition and Pre-Processing

2.1 Hardware

- 32-channel EMOTIV Flex Saline headset.
- Sampling rate $f_s = 128\,\text{Hz}$

3 Spectrogram Cube Construction

A sliding-window short-time Fourier transform (STFT) is applied:

Parameter	**Setting**
Window length L	0.5 s (64 samples)
Hop s	0.125 s (16 samples)
FFT	Software Handled (Block FFT)
Frequency resolution Δf	$1/L = 2\,\text{Hz}$
Slices kept T	10 (covers 1.25 s context)
Output shape	$32 \times F \times 10$ with $F \in \{4, 20, 40\}$

Table 1: Spectrogram parameters used in this project.

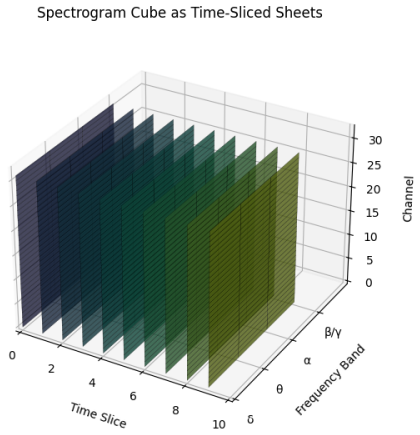

Figure 2: Conceptual spectrogram cube: *rows* = electrodes, *columns* = frequency bins (or 4 band powers), *depth* = recent time slices.

4 3-D Depth-Wise CNN for EEG Spectrogram Cubes

This section explains how we process spectrogram cubes using a 3-D depth-wise convolutional network. Each stage is described with its purpose, tensor shapes, and rationale.

4.1 Input Data Shape

Each training sample arrives as a tensor:

$$X_0 \in R^{B \times C \times F \times T}$$

where:

- $C = 32$: EEG channels (electrodes)
- $F = 5$: frequency bins (5 band powers)
- $T = 10$: short-time windows (e.g. 10 slices of 125 ms each)

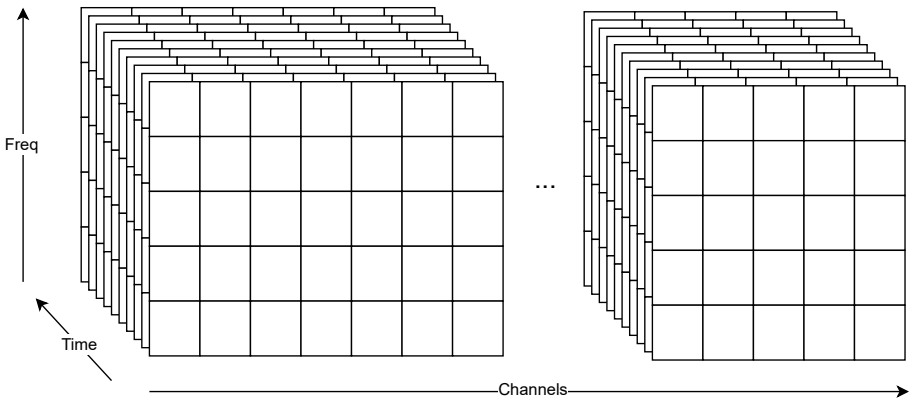

Figure 3: Spectrogram cube input: 32 electrodes × 5 frequency bins × 10 time slices

4.2 Why 3-D CNN?

EEG spectrograms have three spatial axes: electrodes, frequency, and time. A 3-D CNN allows filters to detect patterns that span frequency and time—e.g., "beta power rising for 300 ms." This preserves spectral-temporal structure compared to flattening.

4.3 Depth-Wise vs Full 3-D Kernels

- **Full 3-D**: kernel spans all electrodes—early mixing of signals
- **Depth-Wise (used here)**: kernel size $(1 \times k_F \times k_T)$; each electrode is processed independently

4.4 Layer-by-Layer Architecture

1. Depth-Wise Conv3D (First Layer)

```
Conv3d(
    in_channels=32, out_channels=32,
    kernel_size=(1, 3, 3), padding=(0,1,1), groups=32
)
```

Each electrode gets its own 3×3 sliding filter, applied only over its frequency-time grid.

- Detects local textures like: "beta rise over 300 ms", "8–10 Hz bump"
- Depthwise = no mixing between electrodes yet

2. BatchNorm + ReLU Stabilizes training across small BCI batches.

3. MaxPool3D (1,2,2) Reduces size in frequency and time:

$$F = 5 \to \begin{cases} 2 & \text{no padding (floor)} \\ 3 & \text{with padding or } \texttt{ceil\_mode=True} \end{cases}, \quad T = 10 \to 5$$

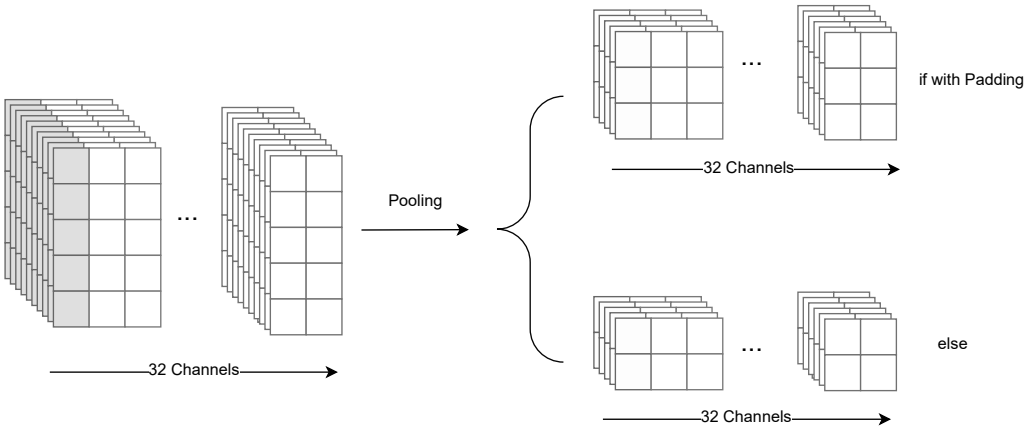

Figure 4: 3D max pooling downsamples each electrode's feature map in frequency and time, preserving strongest activations.

4. Depth-Wise Conv3D (Second Layer) Same kernel config as before, now on pooled maps. We apply two filters (as opposed to one in the first round) this time round and essentially double the number of channels.

- Each electrode now outputs 2 different features.
- Total channels: $32 \times 2 = 64$

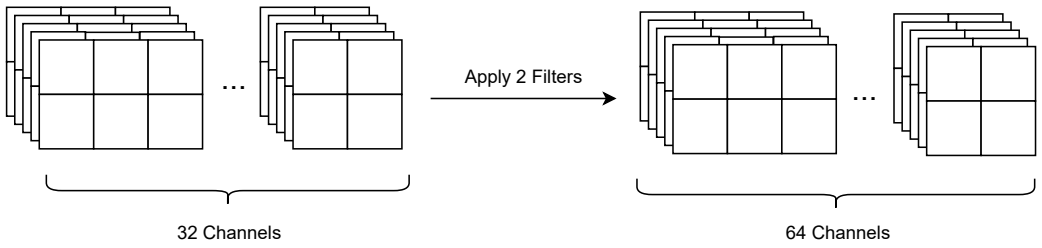

Figure 5: Pointwise (1×1×1) convolution mixes channels at each location to form spatial contrasts and joint features.

5. Pointwise Conv3D (1×1×1)

```
Conv3d(
    in_channels=64, out_channels=64,
    kernel_size=1, groups=1
)
```

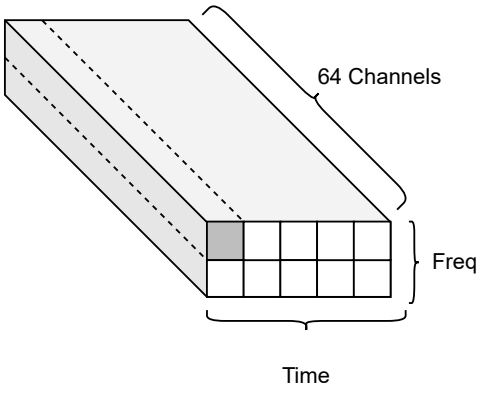

Figure 6: Pointwise (1×1×1) convolution mixes channels at each location to form spatial contrasts and joint features.

Each voxel's 64-D vector is multiplied by a learnable 64×64 matrix.

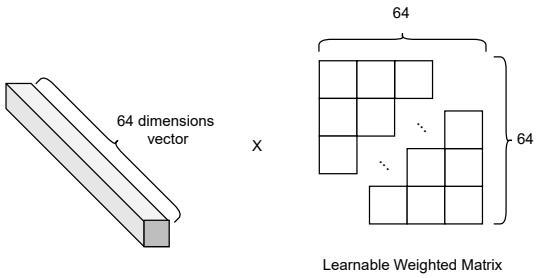

Figure 7: 64 channel feature vector does a pointwise convolution with a learnable weighted matrix.

- Enables combinations like "C3-α minus C4-α"
- Acts like a feature mixer or spatial contrast engine

6. Optional Pool or Dropout

- `MaxPool3D(1,2,1)` can further reduce frequency
- `Dropout3D(p=0.3)` zeroes entire feature maps for regularisation

4.5 Reshaping for the GCN

After convolution:

$$\text{shape} = (B, 64, F' = 3, T' = 5)$$

We split channels back into (electrodes, features):

```
k = 2  # features per electrode
x = x.view(B, 32, k, 2, 5)
x = x.permute(0, 4, 1, 2, 3)  # (B, T', electrodes, k, F')
x = x.reshape(B * T', 32, 2 * 3)  # (B*5, 32, 6)
```

4.6 What Did the CNN Learn?

- Spectral bursts (α, β power changes)
- Short temporal slopes
- Contrasts between electrode pairs via pointwise mixing

4.7 Why Depthwise CNN First?

- EEG noise is channel-specific: better to extract clean local features first
- GCN receives better-structured node features

Mini-summary: *The 3-D depthwise CNN acts as a local texture extractor, turning a noisy spectrogram cube into a clean set of frequency-aware, spatially isolated feature vectors—ideal inputs for GCN and temporal models.*

5 Graph Convolutional Network (GCN)

After the depth-wise 3D CNN has extracted frequency-aware features from each electrode, we now treat the 32 electrodes as nodes in a graph. The goal of the GCN is to propagate and refine these local features by leveraging spatial relationships between electrodes.

5.1 Input to the GCN

At this point, the feature tensor has been reshaped into:

$$\text{shape} = (B \cdot T', 32, k \cdot F')$$

Each of the 32 electrodes (nodes) is represented by a d-dimensional feature vector (e.g., $d = 6$ or $d = 64$ depending on pooling and k).

5.2 Electrode Graph

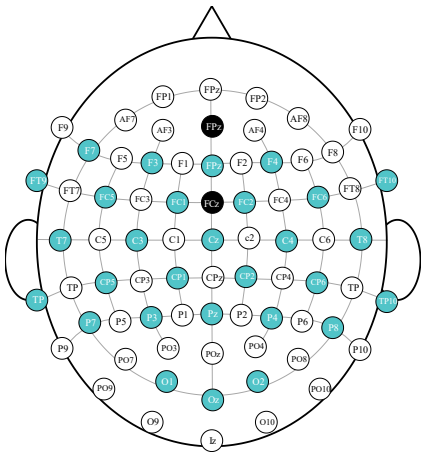

Figure 8: 32 EEG Node Channels represented in a GCN.

We use a static graph based on the 10–20 electrode cap layout:

- Nodes: 32 EEG electrodes
- Edges: connect electrodes that are within a 6 cm radius on the scalp

The adjacency matrix A is normalised with self-loops to become:

$$\tilde{A} = D^{-1/2}(A + I)D^{-1/2}$$

where D is the degree matrix. This ensures stable propagation across the graph.

5.3 Two-Layer GCN Architecture

For each time slice, we apply the following operations:

$$H_1 = \text{ReLU}(\tilde{A}XW_0)$$

$$H_2 = \text{ReLU}(\tilde{A}H_1W_1)$$

Where:

- $X \in R^{32\times d}$ is the feature matrix from the CNN (1 per electrode)
- W_0, W_1 are learnable weight matrices
- $\tilde{A}$ mixes information across neighbouring electrodes
- H_2 is the output feature matrix (e.g., dimension 32×64)

This lets the model learn spatial patterns like:

- "C3 is high while C4 is low" (motor asymmetry)
- "synchronous bursts across frontal sites"

5.4 Why GCN After CNN?

The CNN ensures each electrode has a clean, frequency-rich signal representation. The GCN builds on this by learning how different electrode sites relate at each time step.

It avoids early mixing of noisy raw channels and instead mixes only meaningful, filtered features. This aligns with neuroscientific principles of volume conduction and sensor independence.

6 Temporal Model: TCN *or* Tiny Transformer

6.1 Causal TCN Configuration

Four 1-D causal convolutions with kernel size 5 and dilations 1–2–4–8 (receptive field $\approx 1.5\,\mathrm{s}$).

6.2 Lightweight Transformer Configuration

- 2 encoder layers, 4 heads.
- Model dimension 128, feed-forward dimension 256.
- Causal mask to prevent future leakage.

Both variants run in under 10 ms on a Jetson Orin Nano.

Figure 9: Unrolling of causal TCN receptive field across five spectrogram frames. Grey blocks show dilated convolution reach.

7 Prediction Head and Command Decoding

1. **Fully connected head**: $\mathrm{ReLU}(W_2\,\mathrm{Drop}(\mathrm{ReLU}(W_1 h))) \to$ out.
2. **Classification mode**: softmax over five gestures (*left, right, up, grasp, rest*).
3. **Regression mode**: `tanh` outputs three velocities $(v_x, v_y, v_z) \in [-1, 1]$.

A rule-based post-processor converts labels or velocities into robot-specific joint commands.

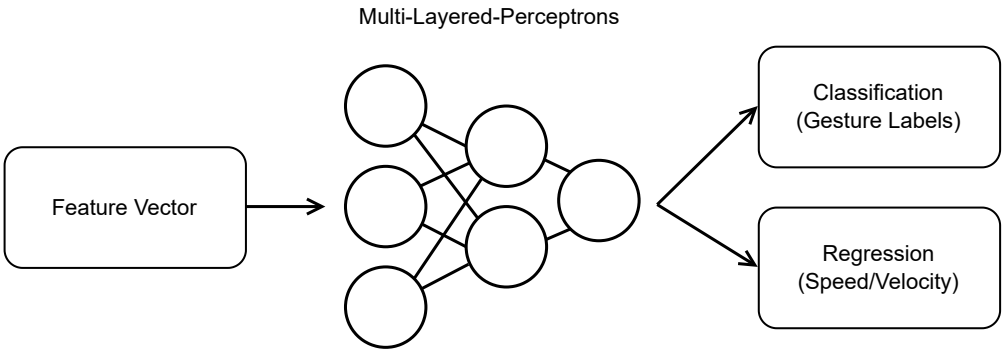

Figure 10: Handoff from transformer(Feature Vector) to Decision Head .

8 Latency and Resolution Trade-Offs

Component	Typical delay (ms)
Data age ($L/2$)	250 (0.5 s window)
Streaming Goertzel filters	< 5
CNN+GCN+TCN inference	10–15
Robot link (CAN/Wi-Fi)	10–20
Total	~ 300

Table 2: Baseline latency budget. Shrinking L or using early-exit cuts the dominant first term.

Aim for a total below 150 ms for joystick-like control; 300 ms is acceptable for discrete actions.

9 Training Details

- Loss: cross-entropy (classification) *or* smooth-L_1 (regression).
- Optimiser: AdamW, $\eta = 5 \times 10^{-4}$, cosine schedule.
- Batch size: 64 windows (~ 10 s of EEG).
- Regularisation: dropout 0.3, weight decay 10^{-4}, early stop on validation loss.

Acknowledgements

To add