-: PPT Presentation :-

By Kushanavo Rakshit, Swagato Chakraborty, Aditya Joshi

Summer Internship

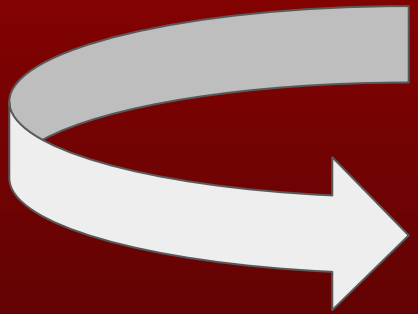
Project supervisor :- Prof Prasoon Kumar
BM dept
NIT Rourkela

Project Topic

Design and development of a Bionic Arm using 4 channel EMG signals

Affordable Bionic Arms: A Project Overview

This presentation outlines the development of a low-cost bionic arm, focusing on the use of electromyography (EMG) signals and machine learning to control the prosthetic. We aim to make this technology accessible to a wider population by reducing the cost to under $200.



Problem Statement and Objective

The current high cost of prosthetic arms, often exceeding $20,000, makes them inaccessible to many amputees, especially in lower-income regions.

Our objective is to develop a myoelectric prosthetic arm with similar capabilities at a significantly lower cost—under $200. By focusing on open-source designs, cost-effective hardware, and efficient software systems, we aim to make bionic prosthetics more affordable and accessible.

-: EXISTING SOLUTIONS AVAILABLE IN MARKET :-

Fig. 2: Currently Available Myoelectric Prosthetics

(a) Michelangelo Hand by Otto Bock ($60,000) {Source: Ottobock US. "Michelangelo prosthetic hand."

(b) i-limb by Touch Bionics (₹1.5L) {Source: Designboom. "Touch Bionics app controlled prosthetic hand."

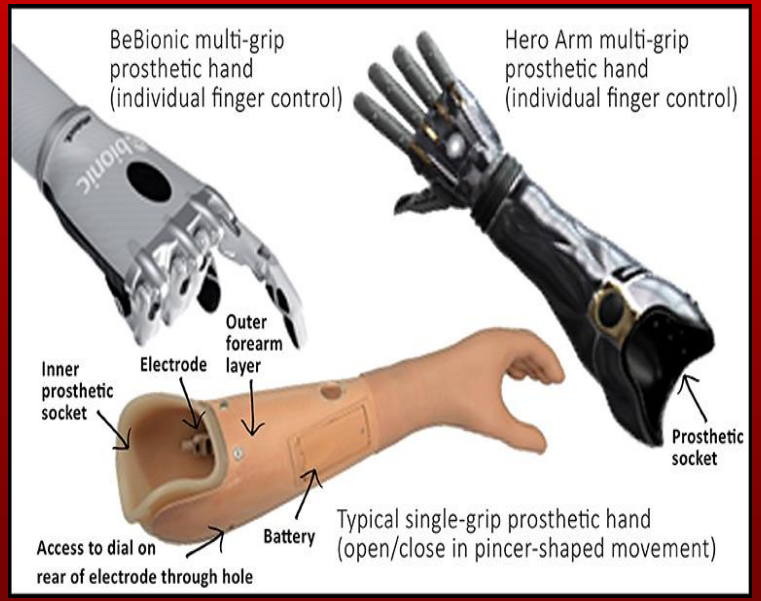
(c) X-limb ($200) {Source: ResearchGate. "X-Limb prosthetic hand with pinch grasp (left) and power grasp."

Literature Review

S.No	Paper	Objective
1	Hand Gestures Recognition for Human-Machine Interfaces: A Low-Power Bio-Inspired Armband	The objective of the research is to develop an arm band and effective signal processing algorithm with Average Threshold Crossing.
2	A bionic hand controlled by hand gesture recognition based on surface EMG signals: A preliminary study	To develop sEMG pattern recognition with the K-nearest value technique.
3	Multi-day dataset of forearm and wrist electromyogram for hand gesture recognition and biometrics	To develop the advanced prosthesis control systems for rehabilitation of upper limb amputees.
4	Surface Electromyographic Signals Using Dry Electrodes	To develop more effective and long-term useable electrodes.

Hardware Components

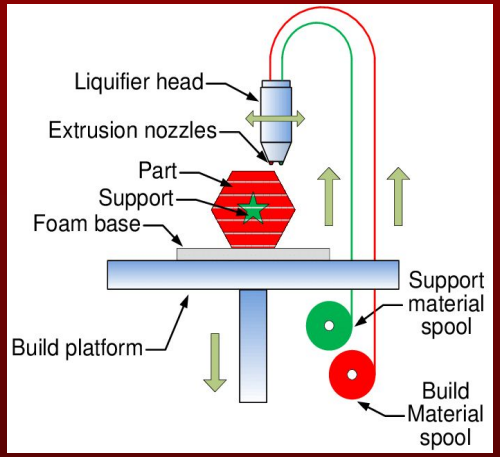
Component	Description / Function
ADC (Analog to Digital Converter)	Converts EMG signals from analog to digital for processing.
Raspberry Pi Zero 2W	Main processing unit for controlling the prosthetic and running algorithms.
SD Card	Stores data for the model and program execution.
Servo Motors	Actuate the mechanical parts of the prosthetic (e.g., fingers).
Muscle Sensor Kit	Detects muscle signals to control the prosthetic.



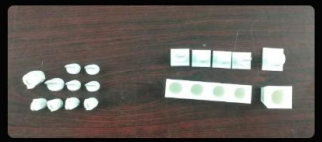
3D Printing

3D printing plays a crucial role in the development of our prosthetic arm. We utilize open-source designs from the **"Inmoov.com"** project to create various parts, including fingers, fingertips, palm region, wrist gears, and pulleys.

This approach allows for customization and flexibility in design, enabling us to create a prosthetic that fits the individual needs of the user.



The parts we have printed and learnt about additive manufacturing are as follows:



FINGER TIPS

PALM COVER AND ADAPTERS

ALL OTHER PARTS

(includes wrist gear, pulleys, palm cover, all the fingers, adapters, palm cover etc)

Methodology

1

EMG Signal Acquisition

Surface electrodes detect muscle signals.

2

Signal Processing

ADC digitizes data for computational analysis.

Noise filtering and signal extraction.

3

ML Pipeline

1D Temporal CNN for feature extracti

MLP for servo angle prediction.

4

Actuation

Servo motors pull the nylon cables causing the fingers to move

3D Printed Prosthetic Hand

Winding Technique

A unique aspect of this prosthetic is the sophisticated winding technique used for tension and actuation. The fingers and hinge joints feature a single winding of thread for smooth and controlled movement. At the final joint near the pulley, a double winding is implemented to enhance stability and grip strength. The tension wire is securely pressed below the adapter region, while the actuation wire is precisely wound around the servo motors, ensuring accurate and reliable control.

-: Materials Design And Principle :-

Cheap Materials :-

PLA for 3D-printed parts (lightweight, cost-effective). Nylon ropes for tension-based actuation, reducing motor requirements. Servo Motors and raspberry pi for actuation

- The basic structure of the hand is 3D printed.
- Despite limited functionality because of lack of strength due to our manufacturing process, a grip force of 21.5N has been achieved by similar methods
- Tension in nylon ropes pulls fingers back to their extended position. Motors control contraction, reducing complexity and cost.

Similar setups have achieved finger flexion speed of 1.3sec, a minimum grasping cycles of 45,000 (while maintaining its original functionality)

What is EMG?

1 EMG Signals

Electromyography (EMG) is a technique used to measure the electrical activity produced by muscles.

2 Action Potentials

Muscle activity generates electrical signals called action potentials (APs), which can be detected by sensors placed on the skin.

3 EMG Output Data

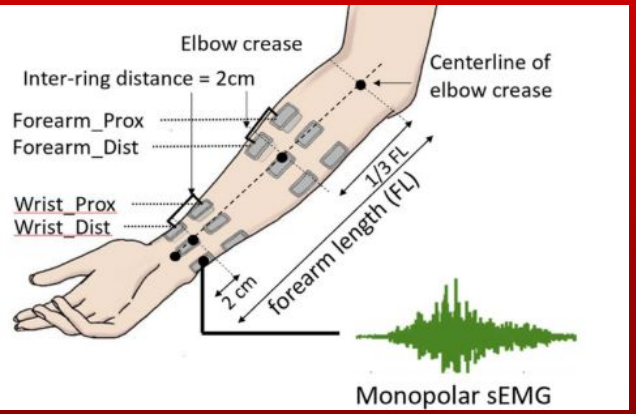
EMG output data is a superimposition of all the APs coming from all the motor units that the electrode is detecting

4 Feature Extraction

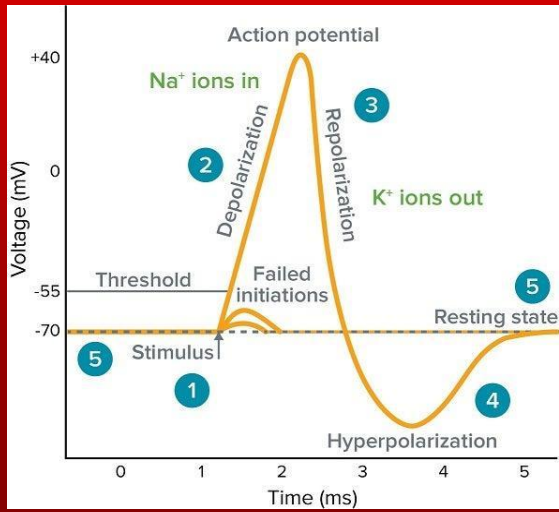
Instead of manual feature extraction, the CNN first detects APs and then patterns in the APs, so features are automatically detected from the output data

From Henneman's law (Motor units are recruited from smallest to largest based on the size of the motor neuron and the associated muscle fibers) we concluded that extracting information about the type of muscle fibers that are firing would give us an insight on the amount of force we need to produce in our arm. We observed that the APs of different muscle fiber types have a certain range of frequencies.

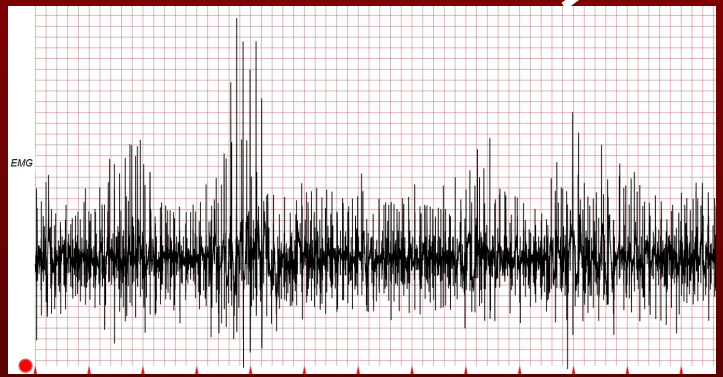
-: Decoding EMG Signals :-



The raw EMG signal is a superimposition of multiple action potentials (APs) of different frequencies and amplitudes generated by nerve communication. These APs are voltage spikes that represent the electrical activity of nerve communication.



The signal is noisy and requires filtering and feature extraction to isolate relevant information. The patterns in the APs, Features of the APs and the location where the signal was taken from will determine the corresponding motion that is to be produced.



As force requirement varies, the APs produced to signal that higher force movement will also vary. Usually, high force activities produce APs with much higher frequencies and amplitudes. [5]

Such features are to be extracted so that we can understand the motion that the amputee was intending to perform

Where Machine Learning Comes in ?

Traditional Methods :-

Traditional feature extraction relies on manually selecting values like root mean square (RMS), mean absolute value (MAV), and zero crossings. These values, measured over time, reveal patterns in muscle activity related to movement.

Convolutional Neural Networks (CNNs) are specialized deep learning algorithms designed for processing grid-like data, such as images or time-series signals. They are particularly effective in recognizing patterns and features through hierarchical layers.

Key Components:

1. **Convolutional Layers**: Apply filters to extract local features.
2. **Pooling Layers**: Reduce spatial dimensions while retaining important information.
3. **Fully Connected Layers**: Integrate extracted features for classification or regression tasks.

CNNs are widely used in applications like image recognition, signal processing (e.g., EMG), and object detection due to their ability to learn features automatically from raw data.

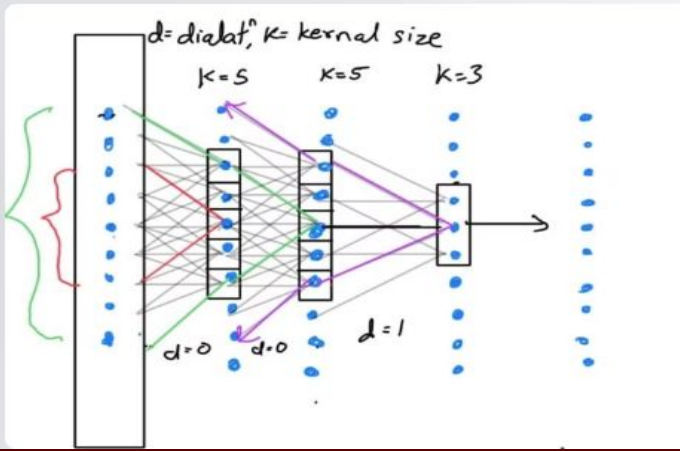
Machine Learning :-

Machine learning algorithms, such as convolutional neural networks (CNNs), offer a more efficient approach. CNNs automatically extract features, identify patterns, and predict movement.

Current research often focuses on recognizing specific gestures; but with sufficient data, CNNs could potentially predict precise servo motor angles, enabling more complex prosthetic control.

Signal Processing Workflow

1. **Signal Acquisition**: EMG signals are captured via electrodes placed on the skin or within muscles.
2. **Preprocessing**: Filters remove noise and amplify relevant signals.
3. **Feature Extraction**: Characteristics like amplitude, frequency, and duration are computed.
4. **Classification**: CNNs analyze features to determine muscle activity patterns.



Detailed Explanation of CNN Layers

1. **Input Layer**: Accepts preprocessed EMG signals.
2. **Convolutional Layers**:
   - Use filters to detect patterns such as spikes in EMG signals.
   - Example: A filter might identify high-frequency components correlating to fast muscle contractions.
3. **Activation Functions**: Apply non-linearity to introduce complexity in feature detection. Commonly used functions include ReLU (Rectified Linear Unit).
4. **Pooling Layers**: Reduce the spatial size of feature maps to focus on the most important elements.
5. **Fully Connected Layers**:
   - Combine all detected features to classify muscle activity.
   - Example: "Grip" vs. "Release" action in prosthetics.

Now we have to define few terms such as:

Threshold Potential: The voltage level that must be reached to trigger an AP.

- **Kernel**: A matrix used in CNNs to extract features from input data.
- **Pooling**: A down-sampling technique to reduce data size while preserving essential features.
- **Dilation**: Expands the receptive field by skipping certain elements in the input.
- **Feature Map**: The output of convolution layers, representing the presence of specific features in the input data.

```python
import numpy as np
import tensorflow as tf
# Discretized AP Arrays (0->+max->0->-max->0)

ap_typeI = np.array([
    0.0,  0.4,  0.9,  1.0,  0.7,  0.2,  0.0, -0.2, -0.5,
   -0.7, -0.5, -0.3, -0.1,  0.0,  0.0
], dtype=np.float32)

ap_typeIIa = np.array([
    0.0,  0.7,  1.0,  0.4,  0.0, -0.3, -0.7, -0.3,  0.0,  0.0
], dtype=np.float32)

ap_typeIIb = np.array([
    0.0,  1.0,  0.3,  0.0, -0.5, -0.2,  0.0
], dtype=np.float32)

ap_silence = np.array([
   -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0
], dtype=np.float32)


# 1) Feature Definitions & Dilation/Padding (First Layer)
sampling_rate = 20000  # 20 kHz

features = [
    {"name": "Type I",   "frequency": 30,  "kernel_size": 15},  # Slow Twitch
    {"name": "Type IIa", "frequency": 100, "kernel_size": 10},  # Medium Twitch
    {"name": "Type IIb", "frequency": 225, "kernel_size": 7 },  # Fast Twitch
    {"name": "Silence",  "frequency": None,"kernel_size": 7 }   # Silence => size=7
]

def compute_dilations_and_padding(feature, sampling_rate, num_layers=4):
    ksize = feature["kernel_size"]
    if feature["frequency"] is None:
        dilation_rates = [1]*num_layers
    else:
```

Importing libraries and discretized AP arrays:-

1. *Purpose*: The code models different types of Action Potential (AP) patterns for biological signals and processes them in a layered fashion using convolutional techniques.
2. *Libraries*:
   - numpy for numerical operations.
   - tensorflow for deep learning tasks.
3. *Action Potentials (AP)*: Arrays represent signal types:
   - *Type I*: Slow-twitch fibers.
   - *Type IIa*: Medium-twitch fibers.
   - *Type IIb*: Fast-twitch fibers.
   - *Silence*: Baseline silence signal.

   These mimic real-world biological signals.
4. *AP Array Shapes*: Each array's size reflects the kernel size of convolution layers that will process these signals.
5. *Data Type*: Defined as float32 for compatibility with TensorFlow operations.

```
        dilation_rates = [d0]
        for _ in range(num_layers-1):
            dilation_rates.append(dilation_rates[-1]*2)
    total_pad = sum((ksize - 1)*d for d in dilation_rates)
    return dilation_rates, total_pad

for feat in features:
    feat["dilation_rates"], feat["total_padding"] = compute_dilations_and_padding(feat, sampling_rate)


# 2) First Layer Processing (4-layers)
def first_layer_processing(input_data, num_channels=4):
    signal_length = input_data.shape[1]
    num_features = len(features)
    channel_list = []
    ap_kernels = {
        "Type I":   tf.keras.initializers.Constant(ap_typeI.reshape((15,1,1))),
        "Type IIa": tf.keras.initializers.Constant(ap_typeIIa.reshape((10,1,1))),
        "Type IIb": tf.keras.initializers.Constant(ap_typeIIb.reshape((7,1,1))),
        "Silence":  tf.keras.initializers.Constant(ap_silence.reshape((7,1,1)))
    }
    for ch in range(num_channels):
        ch_data = input_data[ch,:]
        feat_outputs = []
        for feat in features:
            ksize = feat["kernel_size"]
            dilations = feat["dilation_rates"]
            total_pad = feat["total_padding"]
            feat_name = feat["name"]
            x = tf.reshape(ch_data, (1, signal_length, 1))
            x = tf.pad(x, [[0,0],[total_pad,0],[0,0]], mode="CONSTANT")
            for d in dilations:
                init = ap_kernels[feat_name]
                conv = tf.keras.layers.Conv1D(
                    filters=1,
                    kernel_size=ksize,
```

Feature Definitions and Dilation/padding:

1. *Features*: Defined for each AP type with frequency and kernel size.
    - Frequency reflects the rate of signal repetition.
    - Kernel size defines the window of signal processed in each step.
2. *Sampling Rate*: Signals are sampled at 20,000 Hz (20 kHz), simulating high-resolution biological data.
3. *Dilations*: A method to expand the effective receptive field of the convolution without increasing kernel size, calculated dynamically based on signal frequency.
4. *Padding*: Ensures the input signal retains its dimensions during processing. Total padding is determined based on kernel size and dilation rates.
5. *Dynamic Calculation*: For each feature, dilation rates and padding are precomputed for efficient processing.

```
                    use_bias=False,
                    kernel_initializer=init,
                    trainable=False
                )
                x = conv(x)
            out_1d = tf.squeeze(x).numpy()
            feat_outputs.append(out_1d)
        min_len = min(len(a) for a in feat_outputs)
        feat_outputs = [a[:min_len] for a in feat_outputs]
        stacked = np.stack(feat_outputs, axis=0)
        channel_list.append(stacked)
    min_len_overall = min(x.shape[1] for x in channel_list)
    channel_list = [x[:, :min_len_overall] for x in channel_list]
    first_out = np.stack(channel_list, axis=0)  # => (4,4,time_steps)
    return first_out
# 3) Second Layer with 2 Sub-Layers (dilations=10,20) + Feature Stride
def second_layer_processing_with_strides(first_out, strides_per_feature, num_filters=4):
    kernel_size=10
    second_layer_dilations = [10, 20]
    num_channels, num_features, time_steps = first_out.shape
    second_list = []
    for ch in range(num_channels):
        for j in range(num_features):
            data_1d = first_out[ch, j, :]
            data_3d = tf.reshape(data_1d, (1, time_steps, 1))
            stride = strides_per_feature[j]
            x = data_3d
            for d in second_layer_dilations:
                conv = tf.keras.layers.Conv1D(
                    filters=num_filters,
                    kernel_size=kernel_size,
                    strides=stride,
                    dilation_rate=d,
                    padding="same",
                    activation="relu"
                )
```

First Layer Processing :-

1. *Purpose*: Initial processing of raw input signals across multiple channels (4 channels in this example).
2. *Input Shape*: Signals are reshaped and padded for convolution.
3. *Kernels*: Predefined AP arrays serve as fixed kernels for convolution, simulating signal-specific filters.
4. *4-Layer Convolution*:
    - Each layer applies a convolution with increasing dilation rates.
    - This progressively extracts features at different scales.
5. *Feature Stacking*: Outputs from different features (AP types) are aligned and stacked, ensuring all features have the same length for further processing.

```
    new_times = [second_array[i].shape[1] for i in range(len(second_array))]
    min_nt = min(new_times)
    trimmed = []
    idx = 0
    for ch in range(num_channels):
        for j in range(num_features):
            arr = second_array[idx]
            arr = arr[:, :min_nt, :]
            trimmed.append(arr)
            idx += 1
    trimmed_np = np.array(trimmed)
    trimmed_np = trimmed_np.reshape(num_channels, num_features, 1, min_nt, num_filters)
    final_tensor = np.squeeze(trimmed_np, axis=2).transpose(0, 1, 3, 2)
    return final_tensor

# 4) MLP for Outputs
def build_servo_mlp(input_dim, num_outputs=5):
    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(input_dim,)),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(num_outputs, activation=None)
    ])
    return model


# 5) Full Pipeline with Previous Angles Concatenation
def process_test_data(custom_test_data=None, signal_length=1000, num_channels=4,
                      num_filters=4, prev_angles=None):
    if custom_test_data is None:
        input_data = tf.random.uniform((num_channels, signal_length))
    else:
        input_data = tf.convert_to_tensor(custom_test_data, dtype=tf.float32)

    first_out = first_layer_processing(input_data, num_channels=num_channels)
    second_layer_strides = [1, 2, 3, 1]
```

Second layer Processing with strides :-

1. *Second Layer: Enhances extracted features using dilations and strides.*
   - Strides: Downsample the signal to focus on dominant features.
   - Dilations: Set to 10 and 20, capturing long-range dependencies in signals.

2. *Multiple Filters*: Each convolution in this layer produces multiple filters (default: 4 filters).

3. *3D to 2D Transformation*: Outputs are reshaped and aligned across channels and features.

Output Shape: (Channels × Features × Time Steps × Filters), enabling further processing in higher layers.

```
    flatten_vec = second_out.reshape((1, -1))
    if prev_angles is None:
        prev_angles = np.zeros(5)
    flatten_vec_with_angles = np.concatenate([flatten_vec[0], prev_angles])
    flatten_vec_with_angles = np.expand_dims(flatten_vec_with_angles, axis=0)
    input_dim = flatten_vec_with_angles.shape[1]
    mlp = build_servo_mlp(input_dim=input_dim, num_outputs=5)
    raw_pred = mlp.predict(flatten_vec_with_angles)[0]
    clamped = np.clip(raw_pred, 0, 180)
    print("Predicted angles:", clamped)
    return second_out, clamped

if __name__ == "__main__":
    test_data = np.random.rand(4, 1000).astype(np.float32)
    prev_output = np.array([90, 45, 60, 120, 30])
    second_out, angles = process_test_data(test_data, prev_angles=prev_output)
    print("Final angles:", angles)
```

To enhance the model's ability to interpret APs from the EMG signals, we have predefined a set of specialized kernels in the first convolutional layer. These kernels are tailored to detect APs within specific frequency ranges, ensuring efficient feature extraction from the raw EMG data.

We will employ the freeze-unfreeze training methodology for optimal performance. Initially, the predefined kernels will remain fixed, and we will train only the second convolutional layer along with the Multi-Layer Perceptron (MLP). Once this phase is complete, we will unfreeze the kernels in the first layer and proceed to fine-tune the entire model. This approach ensures that the predefined kernels are both effective at their intended task and adaptable to the dataset, ultimately leading to a more accurate and robust CNN.

Multi layer Perceptron and Full pipeline of the code:

MLP (Multi-Layer Perceptron):
- Converts extracted features into servo control angles.
- Uses three dense layers for feature reduction and mapping to outputs.

1. *Previous Angles*: Concatenates previous servo angles with the processed signal features to add temporal continuity.

2. *Predictions*: MLP predicts raw angles for servos, clamped to the range [0, 180] degrees.

3. *Pipeline Workflow*:
- Input signals → First Layer → Second Layer → Flattened Features → MLP → Predicted Angles.

4. *Testing and Example Usage*: Random test signals simulate input data. Predefined servo angles allow testing the continuity of predictions.

5. *Output*: The predicted servo angles, representing processed signal control, are printed for interpretation.

1D Temporal CNN - Architecture Overview :-

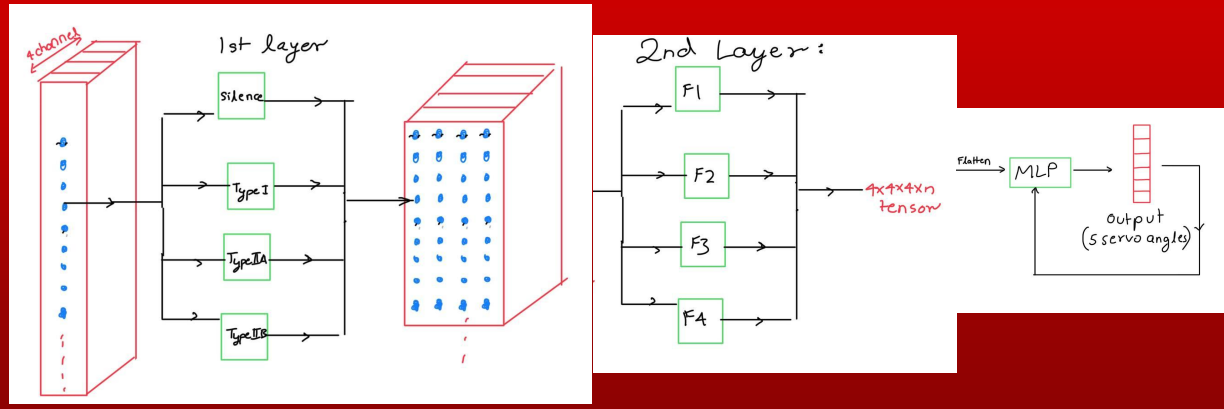
INPUT :- (Raw EMG Data)

1st Layer - Detects APs of different freq ranges

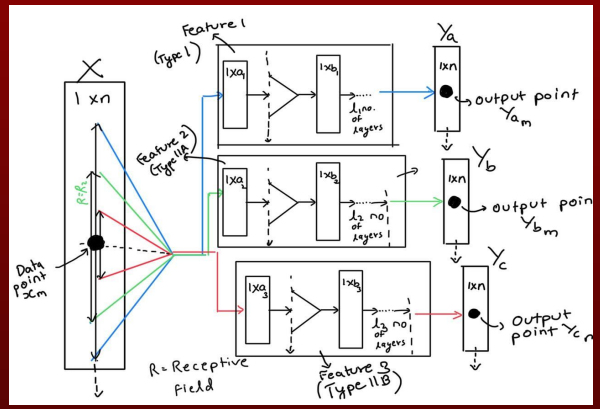
2nd Layer - Recognises patterns in the APs

MLP - Servo Angle Prediction

Output :- (Servo Motor Angles)



Complete 1D Temporal CNN Model for EMG Processing



Detailed View of Layer 1 – Action Potential Detection in the CNN Model

Conclusion :-

Impact

Combining ML and EMG enables real-time, intuitive prosthetic control.
The system adapts to individual users while remaining affordable.

Real time control with cost effectiveness

Gripping various objects (e.g., spheres, rectangles) to test strength and flexibility.

Validation Setup (Gripping Various Objects for Strength & Precision Testing)

Future Work

- Collect and analyze EMG data.
  Modify and Test the model for other prosthetic limbs
- Addition of feedback mechanisms such as FSRs to detect slipping while holding objects, and prevent it.
- Addition of Haptic feedback
- Reduction of computational load on inbuilt electronics by connecting the prosthetic to a mobile phone to perform complex computation without needing powerful microprocessors

References

[1] Mohammadi, A., Lavranos, J., Zhou, H., Mutlu, R., Alici, G., Tan, Y., Choong, P. and Oetomo, D., 2020. A practical 3D-printed soft robotic prosthetic hand with multi-articulating capabilities. *PloS one*, ***15*(5)**, pp.e0232766.

[2] Gentile, C. and Gruppioni, E., 2023. A perspective on prosthetic hands control: From the brain to the hand. *Prosthesis*, ***5*(4)**, pp.1184-1205.

[3] Pradhan, A., He, J. and Jiang, N., 2022. Multi-day dataset of forearm and wrist electromyogram for hand gesture recognition and biometrics. *Scientific data*, ***9*(1)**, pp.733.

[4] Mongardi, A., Rossi, F., Prestia, A., Ros, P.M., Roch, M.R., Martina, M. and Demarchi, D., 2022. Hand gestures recognition for human-machine interfaces: A low-power bio-inspired armband. *IEEE Transactions on Biomedical Circuits and Systems*, ***16*(6)**, pp.1348-1365.

[5] https://info.tmsi.com/blog/what-is-emg, 29/1/2025, 5:44pm

[6] Phienthrakul, T., 2018, January. Armband gesture recognition on electromyography signal for virtual control. In *2018 10th International Conference on Knowledge and Smart Technology (KST)* (pp. 149-153). IEEE.

[7] Woźniak, M., Pomykalski, P., Sielski, D., Grudzień, K., Paluch, N. and Chaniecki, Z., 2018, September. Exploring EMG gesture recognition-interactive armband for audio playback control. In *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)* (pp. 919-923). IEEE.

[8] Amor, A.B.H., El Ghoul, O. and Jemni, M., 2019, December. Sign language handshape recognition using Myo Armband. In *2019 7th International conference on ICT & Accessibility (ICTA)* (pp. 1-5). IEEE.

[9] Kim, S., Kim, J., Ahn, S. and Kim, Y., 2018. Finger language recognition based on ensemble artificial neural network learning using armband EMG sensors. *Technology and Health Care*, ***26*(S1)**, pp.249-258.

Left to Right :-

Prof Prasoon Kumar
Kushanavo Rakshit
Swagato Chakraborty
Aditya Joshi

Thank You