Wrist Rehabilitation System for CTS Patients

Kothavari Nitya
Artificial Intelligence and Data Science in Medical Engineering
Amrita Vishwa Vidhyapeetham, Coimbatore, India.
cb.ai.u4aim24123@cb.students.amrita.edu

Battari Pavani Shreeya
Artificial Intelligence and Data Science in Medical Engineering
Amrita Vishwa Vidhyapeetham, Coimbatore, India.
cb.ai.u4aim24106@cb.students.amrita.edu

Ardhra Vinod
Artificial Intelligence and Data Science in Medical Engineering
Amrita Vishwa Vidhyapeetham, Coimbatore, India.
cb.ai.u4aim24105@cb.students.amrita.edu

Harikrishna Sivanand IYER
Artificial Intelligence and Data Science in Medical Engineering
Amrita Vishwa Vidhyapeetham, Coimbatore, India.
cb.ai.u4aim24114@cb.students.amrita.edu

***Abstract*—Carpal Tunnel Syndrome(CTS) is a condition that affects the functionality of the wrist, caused by compression of the median nerve within the carpal tunnel of the wrist. In this paper, we present a wearable wrist rehabilitation device that detects whether a person has CTS or not through a trained Deep learning model of Recurrent Neural Network with Long Short-Term memory model and displays the result with the help of an OLED. It also provides therapy with the help of a Servo motor.**

I. INTRODUCTION

Carpal Tunnel Syndrome (CTS) results from compromise of median nerve function at the wrist caused by increased pressure in the carpal tunnel, an anatomical compartment bounded by the bones of the carpus and the traverse carpal ligament. Simply it is a prevalent neuromuscular disorder resulting from the compression of median nerve in the wrist, often leading to numbness, tingling, and weakness in hand. CTS affects mainly middle aged woman. In the majority the exact cause and pathogenesis of CTS is unclear. Early detection and rehabilitative treatment in the early stages could be crucial for the best patient outcome. Yet, traditional CTS assessment methods are subjective, time-consuming, and require professional clinical supervision. Therefore, this study presents a novel, inclusive, intelligent wearable system that can assess CTS and facilitate therapeutic gestures via surface electromyography (sEMG) signal collection.

The contribution of this study is real-time classification of CTS severity levels by a deep learning model based on a recurrent neural network (RNN) with Long Short-Term Memory (LSTM) model is trained on time-series EMG data. The trained model in TFLite form for inference across any modality is sent to a cloud server (via Flask), while the local PC runs a Python script that extracts the trained data in normalized form. The cloud server sends back the classification prediction along with the calculated wrist angle for therapy to the locally-based Arduino system. A low-cost wearable system with Arduino UNO, EMG sensor, DS5160 high torque servo, and an OLED for real-time visualization of CTS severity and wrist movements for non-invasive treatment works as intended. The EMG signal is collected via analog pin and transmitted via serial to the associated PC after assessed with a sliding window approach. Relative to the severity of No CTS, Mild, Moderate, or Severe, received on the PC side, the servo obtains the smooth-calibrated wrist movement at a therapeutically relevant angle (+5° from the resting position).

Our wearable system provides the following benefits:

- A low-cost wearable prototype for EMG-based CTS diagnosis and treatment.
- A real-time prediction pipeline using a sliding window of EMG data and TFlite RNN model stored on a local cloud server.
- A feedback loop mechanism that converts diagnostic output into physical wrist therapy via servo actuation.

This paper details the creation, training, and validation of the deep learning classifier, hardware and cloud communication across systems, and therapeutic movement in real time, which was validated via wrist EMG datasets with known ground truth of CTS severity.

II. Literature Review

A. "Deep Learning-Based Approaches for Enhanced Diagnosis and Comprehensive Understanding of Carpal Tunnel Syndrome"

- This study explains the classification of wrist movements has been done using sEMG signals and a Convolutional Neural Network (CNN) model. The system segregates EMG data into time windows, extracts features, and uses CNN to classify four motions of the wrist. Average classification accuracy was 96.57%. Drawbacks include potential variability in sEMG signals due to sensor placement, muscle fatigue, and inter-subject differences. The model may also require recalibration for different users.

B. "Diagnosis of Carpal Tunnel Syndrome from Thermal Images Using Artificial Neural Networks"

- In this paper,diagnosing CTS is done using thermal imaging and Artificial Neural Networks(ANN). Thermal images were taken from fingers of CTS patients and healthy patients and analyzed. Statistics such as mean temperature change served as inputs for the feed-forward backpropagation neural networks. This model has achieved 93.75% accuracy, 92.86% sensitivity, and 94.44% specificity,indicating it to be a robust model. Minor drawbacks include possible influence from ambient conditions and limited generalization.

C. "A New Approach to Applying Feedforward Neural Networks to the Prediction of Musculoskeletal Disorder Risk"

- This human wrist model includes 23 degrees of freedom for 43 muscles, with detailed representations of intrinsic and extrinsic muscle tendons. Testing and validation were performed by comparing simulated fingertip forces with experimental data. The model aims to support research on hand function, prosthetics, rehabilitation, and ergonomic design.

D. "Classification of the Angular Position During Wrist Flexion-Extension Based on EMG Signals"

- A system was proposed to classify wrist flexion-extension angles using sEMG segmented using an Artificial Neural Network(ANN). 6 angular positions were classified with 95.8% accuracy and 0.0123 mean squared error(MSE),showing strong predictive performance.Drawbacks include dependence on precise electrode placement, muscle fatigue,and inter-subject variations.

E. "A Wearable Sensor System for Monitoring Wrist Posture and Activity"

- Classifications of postures and activities using heel acceleration and plantar pressure data were done with SVM across six classes,achieving around 95.2% average accuracy (full sensor set) and over 98% accuracy (optimized sensor set) 93% accuracy (1 Hz). One drawback is limited validation on only nine subjects and potential variability due to differences in gait or footwear across broader populations.

F. "Comparative Study of Wearable Devices for Detection, Diagnosis, and Rehabilitation of Carpal Tunnel Syndrome"

It proposes a system that integrates biosensors and inertial measurement units (IMUs) to monitor wrist and forearm angles, enabling non-invasive, long-term tracking. The study highlights the use of statistical tools like Wilk and T-tests for data analysis and suggests improvements in existing systems for better tracking and therapeutic applications. While the proposed system shows promise in aiding CTS management, it lacks clinical validation and real-world testing, which limits its immediate applicability.

G. "The Development of Wrist Joint Rehabilitation with Servo Motor Drive for Stroke Handed"

- The study proposed a wrist-hand motion recognition system using sEMG signals processed with SVM, KNN, and decision tree classifiers. 4-channel EMG sensors were segmented into 200 ms windows with 50% overlap, with features like RMS, MAV,and WL. SVM achieved 98.57% accuracy,KNN at 97.14% and DT at 95.71%. Some drawbacks were that performance decreased when tested on new subjects,resulting in poor generalization and variations and the system lacked real-time validation and practical deployment insights.

H. "Effect of Surface Electromyography Electrode Position during Wrist Extension and Flexion Based on Time and Frequency Domain Analyses"

- This paper explains different sEMG electrode positions affect the classification accuracy of wrist flexion and extension movements using time-domain features. Four time-domain features (MAV, RMS, VAR, WL) were classified using Linear Discriminant Analysis (LDA). Average accuracy was 94.5%. The main drawback is limited generalizing due to a small subject pool and reliance on only linear classification and time-domain features.

I. "Recommended Surface EMG Electrode Position for Wrist Extension and Flexion"

- This study presents a technique to indicate better electrode positions for surface EMG of the upper limb muscles during wrist extension and flexion. Three different electrode positions were taken with a 2 cm internal distance between the investigated bipolar electrodes. Accuracy was around 94.2%, with a major drawback being limited generalizability due to testing on a small,specific sample of healthy subjects,

J. "The eWrist- A Wearable Wrist Exoskeleton with sEMG-based Force Control for Stroke Rehabilitation"

- The paper presents a wearable wrist exoskeleton designed for stroke rehabilitation using surface EMG (sEMG) control. It achieves 90% accuracy in classifying wrist flexion and extension tasks. The system uses a nonlinear force controller with sEMG input to provide adaptive support during therapy. The major drawback was limited subject diversity (tested on only 4 subjects) and potential challenges in adapting the system to stroke patients.

III. Proposed Work

We conducted the work in six phases to assess the severity of CTS using real-time EMG signals and provide therapy according to the individual's wrist movement.

A. Phase 1: Data Collection

- We collected Surface Electromyography (sEMG) signals by placing the gel electrodes on the Extensor Carpi Radialis (ECR) muscle once at 0°(relaxed position) and the other at 90°(extended position). In total 100 readings were collected at each position, and a dataset of 1000 samples was linearly interpolated for all wrist angles between 0° and 90°, with the corresponding EMG value along with the CTS severity labels based on clinical thresholds. The four labels were created in the following manner:
  - Angle $> 35^\circ \rightarrow$ No CTS
  - 26°–$35^\circ \rightarrow$ Mild CTS
  - 16°–$25^\circ \rightarrow$ Moderate CTS
  - $\leq 15^\circ \rightarrow$ Severe CTS

  This is a time series dataset that contains EMG values, corresponding angles, and CTS severity labels.
  - The positions of the EMG electrodes for extracting the electrical muscle activity of ECR (Extensor Carpi Radialis). This muscle, located in the forearm, is responsible for extending (straightening) the wrist. So for these signals, the first electrode is placed over the ECR muscle, which is the origin. On the same line, the second electrode is placed 2-3 cm away from the origin. The third electrode is the reference electrode, is placed over the elbow (or any bony area of the hand near the origin).

B. Phase 2: Recurrent Neural Network (RNN) Model and Long Short-Term Memory (LSTM) Model

- RNN Model: The preprocessing of the dataset was done by normalizing the acquired EMG values and creating sequences of 10 readings to reshape the input as a time series for RNN. Then the CTS labels were one-hot encoded for the prediction. To build our model, we used TensorFlow and Keras libraries in Python to use inbuilt functions like SimpleRNN layer, hidden layers, and output layers to get the prediction. The model was trained and evaluated on cross-entropy loss. We tuned the hyperparameters, like changing the sequence length, learning rate, batch size, epochs, number of neurons, activation function, and dropout rate, to find the optimal hyper parameters so that the model is well trained and gives accurate predictions.

  LSTM Model: The preprocessing of the dataset was done the exact same way as RNN model transforming the data into a suitable input format for LSTM networks. CTS labels were one-hot encoded to enable multiclass classification.TensorFlow and Keras libraries was used to built the model, employing the LSTM layer to capture long-term dependencies in muscle signal patterns. We added dense layers for deeper learning and used softmax activation in the output layer for classification. The model was trained and validated using cross-entropy loss. We tuned hyperparameters including sequence length, batch size, number of LSTM units, learning rate, activation functions, epochs, and dropout rate to improve performance and prediction accuracy for CTS detection.

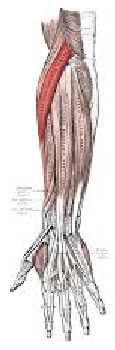


Fig. 1. Extensor Carpi Radialis

C. Phase 3: Cloud Server Interfacing

The trained RNN model is converted to a TensorFlow Lite model to make it compatible for running on microcontrollers. Real-time EMG signals are read from Arduino through serial communication and then sent to a Flask-based cloud server. The Flask API receives a sequence of 10 EMG values and uses the converted TensorFlow Lite (TFLite) model to predict the severity of CTS, returning the corresponding labels and confidence levels. It is then deployed on a PC.

D. Phase 4: Prototype design and 3D printing

In order to provide therapy for people detected with CTS we designed a 3D exoskeleton model. The exoskeleton was designed using Fusion 360. The design includes a semi-open wrist brace with place for the DS5160 servo motor, a flexible platform for Arduino and EMG sensor placement. Further, Servo shaft is connected to palm gear using ball bearing. Polyethylene Terephthalate Glycol(PETG) was used for 3D printing the prototype due to its balance of strength, flexibility and duration.

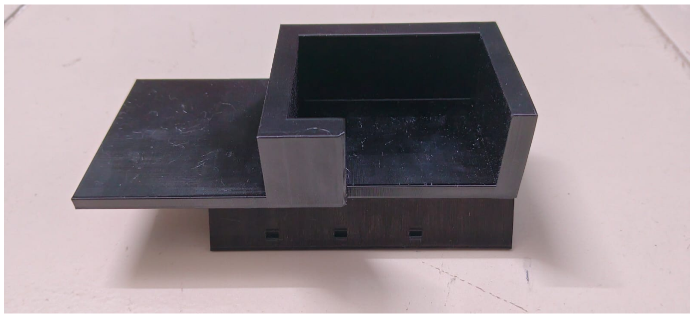


Figure 3.1 Wrist brace

E. *Phase 5: Hardware Integration*

Components Used:

1: EMG Sensor: Captures muscle electrical activity.
2: Arduino UNO Microcontroller: Used to receive data, control the servo motor, and display output.
3: OLED Display (SSD1306): Used to visually display the CTS prediction result.
4: Servo Motor DS516: Provides the actuation for wrist movement therapy.
5: Power Supply: Powers the Arduino and components.
6: Jumper Wires: For connections between components.
Circuit Connections:

1) EMG sensor:
   SIG $\rightarrow$ A0
   GND $\rightarrow$ GND
   Vs+ $\rightarrow$ 9V(1st battery)
   Vs- $\rightarrow$ GND
   GND $\rightarrow$ Midpoint of conection between 9V and GND of the two batteries
2) OLED Display:
   VCC $\rightarrow$ 5V
   GND $\rightarrow$ GND
   SDA $\rightarrow$ A4 (Arduino UNO)
   SCL $\rightarrow$ A5 (Arduino UNO)
3) Servo Motor:
   Signal $\rightarrow$ Digital pin
   VCC $\rightarrow$ External power source
   GND $\rightarrow$ Common GND with Arduino

The above mentioned connections and setup is now capable of reading the EMG data, predicting CTS condition using the Cloud based Rnn model and then providing therapy for the needed.

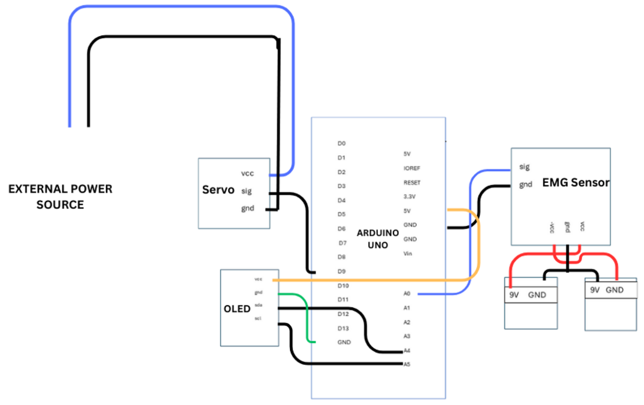


Figure 3.2 Schematic Diagram

F. *Phase 6: Actuation (Therapy)*

Once the EMG values are read it is sent to the cloud server where the values are classified using the RNN model which will predict wrist movement angle. According to the given threshold values if the individual is detected with CTS the therapy is given in such a way that along with the wrist movement angle +5° angle is added for the therapy. Then this angle is sent to Arduino through the serial communication. The Arduino receives the angle and then it will trigger servo motor to move the wrist to the calculated therapeutic angle.

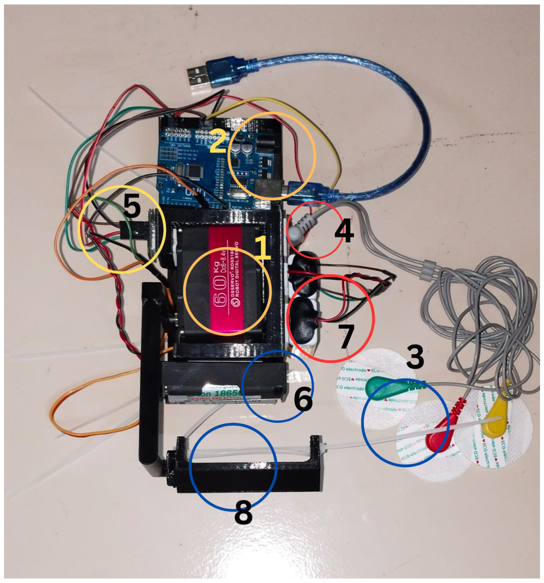


Figure 3.3 1. 3D-printed base holding the servo motor, 2. Arduino microcontroller, 3. EMG sensor electrodes, 4.EMG Micro controller, 5. OLED Display, 6. Battery Holder, 7. Two 9V Lithium ion batteries, 8. Palm brace.

IV. Results and Comparison

The system correctly acquired EMG data in real time, transferred it to the cloud-hosted RNN model for CTS prediction, and the prediction (No CTS / Mild / Moderate / Severe) is sent to Arduino. The result was displayed on the OLED screen connected to Arduino. At the same time, the system calculated the wrist angle from the EMG value, added +5° for therapy, and triggered the servo motor to provide the wrist therapy completing a fully integrated detection and therapy loop.

TABLE I
PERFORMANCE COMPARISON

Feature	RNN Model	LSTM Model
Model Architecture	Simple RNN	LSTM
Hidden Layers	1 RNN + 3 Dense	2 LSTM + 2 Dense
Activation Functions	tanh,softmax	tanh,relu, softmax
Train Accuracy	0.9785	0.9937
Validation Accuracy	0.9798	0.9899
Test Accuracy	0.9697	0.9798
Overfitting Risk	Higher	Lower (due to dropout)
Dropout Used	0.1	Yes (0.1 after each LSTM)
Epochs	15	15

INFERENCE: The LSTM model performed much better than RNN model in terms of accuracy, generalization making it better suitable for making the predictions than the RNN model.

V. CONCLUSION

The proposed system demonstrates a real-time, compact, wearable wrist device for CTS detection and rehabilitation by acquiring live EMG signals from surface EMG electrodes. This work showcases an integration of data acquisition, cloud server interface, real-time display on OLED, and actuation for home-based physical therapy. Future work includes improving the prototype design and therapy mechanism.

REFERENCES

[1] L. Xiloyannis *et al.*, "The eWrist: A wearable wrist exoskeleton with sEMG-based force control for stroke rehabilitation," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 3010–3017, 2018.

[2] J. J. Caballero, J. A. Corrales, and G. S. Martín, "Classification of the angular position during wrist flexion-extension using surface electromyography," *Sensors*, vol. 18, no. 4, pp. 1–14, 2018.

[3] T. Yokoyama, M. Akiyama, and Y. Nishimoto, "Recommended surface EMG electrode position for wrist extension and flexion," *Journal of Electromyography and Kinesiology*, vol. 25, no. 4, pp. 683–688, 2015.

[4] K. Wang *et al.*, "Effect of surface electromyography electrode position during wrist extension and flexion based on time and frequency analysis," *Journal of Healthcare Engineering*, vol. 2018, Article ID 7913023, pp. 1–10, 2018.

[5] F. J. Martínez-Martí, C. M. Travieso, J. B. Alonso, and M. A. Ferrer, "Diagnosis of Carpal Tunnel Syndrome from thermal images using artificial neural networks," *Sensors*, vol. 22, no. 17, pp. 1–14, 2022.

[6] M. J. D. Martínez *et al.*, "Multi-stage classification approach for hand gesture recognition using EMG signals," *Diagnostics*, vol. 13, no. 18, p. 3211, 2023.

[7] N. Gautam and A. Shrivastava, "Comparative Study of Wearable Devices for Detection, Diagnosis, and Rehabilitation of Carpal Tunnel Syndrome," *NeuroQuantology*, vol. 20, no. 22, pp. 1852–1860, 2022.

[8] S. K. Lee, M.-J. Kim, J. Lee, J. H. Lee, and Y. Kwon, "EMG-Based Pattern Recognition for Upper-Limb Motion Intent Prediction: A Feasibility Study," *Diagnostics*, vol. 13, no. 20, p. 3211, 2023. doi: 10.3390/diagnostics13203211

[9] T. A. Kuiken, M. M. Lowery, and N. S. Stoykov, "A comparison of surface and intramuscular myoelectric signal classification," *Journal of Biomechanics*, vol. 33, no. 4, pp. 511–516, 1999. doi: 10.1016/S0003-6870(99)00055-1

[10] Ilbay, K., Übeyli, E.D., Ilbay, G. et al. Recurrent Neural Networks for Diagnosis of Carpal Tunnel Syndrome Using Electrophysiologic Findings. J Med Syst 34, 643–650 (2010). https://doi.org/10.1007/s10916-009-9277-6

APPENDIX

APPENDIX A: SOURCE CODES

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN,
    Dense, Dropout
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt

file_path =
    r"C:\Users\shari\Downloads\interpolated_emg_dataset_with_la
df = pd.read_csv(file_path)
print(df.head())

scaler = MinMaxScaler()
df['EMG'] = scaler.fit_transform(df[['EMG']])

def create_sequences(data, labels,
    sequence_length=10):
    X, y = [], []
    for i in range(len(data) - sequence_length):
        X.append(data[i:i+sequence_length])
        y.append(labels[i + sequence_length])
    return np.array(X), np.array(y)

sequence_length = 10
emg_values = df['EMG'].values
cts_labels = df['CTS_Label'].values

X, y = create_sequences(emg_values, cts_labels,
    sequence_length)

y_encoded = to_categorical(y, num_classes=4)

X_train, X_temp, y_train, y_temp =
    train_test_split(X, y_encoded, test_size=0.2,
    random_state=42)
X_val, X_test, y_val, y_test =
    train_test_split(X_temp, y_temp, test_size=0.5,
    random_state=42)

model = Sequential([
    SimpleRNN(64, input_shape=(sequence_length, 1),
        activation='tanh'),
    Dropout(0.1),
    Dense(64, activation=’relu’),
    Dropout(0.1),
    Dense(64,activation=’relu’),
    Dropout(0.1),
    Dense(4, activation=’softmax’)

        # 4 classes: No CTS, Mild, Moderate, Severe
])
model.compile(optimizer=’adam’,
    loss=’categorical_crossentropy’,
    metrics=[’accuracy’])

history = model.fit(
    X_train.reshape(-1, sequence_length, 1),
    y_train,
    validation_data=(X_val.reshape(-1,
        sequence_length, 1), y_val),
    epochs=15,
    batch_size=32,
    verbose=1)

train_loss, train_acc =
    model.evaluate(X_train.reshape(-1,
    sequence_length, 1), y_train, verbose=0)
val_loss, val_acc =
    model.evaluate(X_val.reshape(-1,
    sequence_length, 1), y_val, verbose=0)
test_loss, test_acc =
    model.evaluate(X_test.reshape(-1,
    sequence_length, 1), y_test, verbose=0)

print(f"\nTrain Accuracy: {train_acc:.4f}, Train
    Loss: {train_loss:.4f}")
print(f"Validation Accuracy: {val_acc:.4f},
    Validation Loss: {val_loss:.4f}")
print(f"Test Accuracy: {test_acc:.4f}, Test Loss:
    {test_loss:.4f}")

model.save("RNN_model.h5")
print("RNN model saved as RNN_model.h5")

plt.figure(figsize=(8, 5))
plt.plot(history.history[’loss’], label=’Training
    Loss’, marker=’o’)
plt.plot(history.history[’val_loss’],
    label=’Validation Loss’, marker=’s’)
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training and Validation Loss vs Epochs")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

from sklearn.metrics import confusion_matrix,
    classification_report, accuracy_score,
    precision_score, recall_score, f1_score
import seaborn as sns
import matplotlib.pyplot as plt

y_pred = model.predict(X_test.reshape(-1,
    sequence_length, 1))
y_pred_classes = np.argmax(y_pred, axis=1)  labels
y_true = np.argmax(y_test, axis=1)

accuracy = accuracy_score(y_true, y_pred_classes)
precision = precision_score(y_true, y_pred_classes,
    average=’weighted’)
recall = recall_score(y_true, y_pred_classes,
    average=’weighted’)
f1 = f1_score(y_true, y_pred_classes,
    average=’weighted’)

conf_matrix = confusion_matrix(y_true,
    y_pred_classes)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d",
    cmap="Blues", xticklabels=[0, 1, 2, 3],
    yticklabels=[0, 1, 2, 3])
plt.title("Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.tight_layout()
plt.show()

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")

print("\nClassification Report:")
print(classification_report(y_true, y_pred_classes))
```

Listing 1. RNN Model

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense,
    Dropout
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt


file_path = r"C:\Users\shari\Downloads \n
            \interpolated_emg_dataset_with_labels.csv"
df = pd.read_csv(file_path)


scaler = MinMaxScaler()
df[’EMG’] = scaler.fit_transform(df[[’EMG’]])


def create_sequences(data, labels,
    sequence_length=10):
    X, y = [], []
    for i in range(len(data) - sequence_length):
        X.append(data[i:i+sequence_length])
        y.append(labels[i + sequence_length])
    return np.array(X), np.array(y)

sequence_length = 10
emg_values = df[’EMG’].values
cts_labels = df[’CTS_Label’].values

X, y = create_sequences(emg_values, cts_labels,
    sequence_length)


y_encoded = to_categorical(y, num_classes=4)


X_train, X_temp, y_train, y_temp =
    train_test_split(X, y_encoded, test_size=0.2,
    random_state=42)
X_val, X_test, y_val, y_test =
    train_test_split(X_temp, y_temp, test_size=0.5,
    random_state=42)


model = Sequential([
    LSTM(64, input_shape=(sequence_length, 1),
        activation=’tanh’, return_sequences=True),
    Dropout(0.1),
    LSTM(64, activation='tanh'),
    Dropout(0.1),
    Dense(64, activation='relu'),
    Dense(4, activation='softmax')
])

model.compile(optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy'])


history = model.fit(
    X_train.reshape(-1, sequence_length, 1),
    y_train,
    validation_data=(X_val.reshape(-1,
        sequence_length, 1), y_val),
    epochs=15,
    batch_size=32,
    verbose=1
)


train_loss, train_acc =
    model.evaluate(X_train.reshape(-1,
    sequence_length, 1), y_train)
val_loss, val_acc =
    model.evaluate(X_val.reshape(-1,
    sequence_length, 1), y_val)
test_loss, test_acc =
    model.evaluate(X_test.reshape(-1,
    sequence_length, 1), y_test)

print(f"\nTrain Accuracy: {train_acc:.4f}, Train
    Loss: {train_loss:.4f}")
print(f"Validation Accuracy: {val_acc:.4f},
    Validation Loss: {val_loss:.4f}")
print(f"Test Accuracy: {test_acc:.4f}, Test Loss:
    {test_loss:.4f}")


plt.figure(figsize=(8, 5))
plt.plot(history.history['loss'], label='Training
    Loss')
plt.plot(history.history['val_loss'],
    label='Validation Loss')
plt.title('Training vs Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()

plt.figure(figsize=(8, 5))
plt.plot(history.history['accuracy'],
    label='Training Accuracy')
plt.plot(history.history['val_accuracy'],
    label='Validation Accuracy')
plt.title('Training vs Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()

from sklearn.metrics import confusion_matrix,
    classification_report, accuracy_score,
    precision_score, recall_score, f1_score
import seaborn as sns
import matplotlib.pyplot as plt

y_pred = model.predict(X_test.reshape(-1,
    sequence_length, 1))
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)


accuracy = accuracy_score(y_true, y_pred_classes)
precision = precision_score(y_true, y_pred_classes,
    average='weighted')
recall = recall_score(y_true, y_pred_classes,
    average='weighted')
f1 = f1_score(y_true, y_pred_classes,
    average='weighted')


conf_matrix = confusion_matrix(y_true,
    y_pred_classes)


plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d",
    cmap="Blues", xticklabels=[0, 1, 2, 3],
    yticklabels=[0, 1, 2, 3])
plt.title("Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.tight_layout()
plt.show()


print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")


print("\nClassification Report:")
print(classification_report(y_true, y_pred_classes))
```

Listing 2. LSTM Model

```
import tensorflow as tf
from keras.saving import load_model


saved_model = "LSTM_model.h5"
model = load_model(saved_model)


converter =
    tf.lite.TFLiteConverter.from_keras_model(model)
converter.target_spec.supported_ops = [
    tf.lite.OpsSet.TFLITE_BUILTINS,
    tf.lite.OpsSet.SELECT_TF_OPS
]


tflite_model = converter.convert()


tflite_converted_model = "LSTM_model.tflite"
with open(tflite_converted_model, "wb") as f:
    f.write(tflite_model)

print(f"TFLite model successfully converted and
    saved to {tflite_conerted_model}")
```

Listing 3. Python Code to convert into TFLite model

```
from flask import Flask, request, jsonify
import numpy as np
import tensorflow as tf

app = Flask(__name__)

```

```python
interpreter = tf.lite.Interpreter(model="LSTM_model.tflite")
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

class_labels = ["No CTS", "Mild CTS", "Moderate CTS", "Severe CTS"]

@app.route('/predict', methods=['POST'])
def predict():
    try:
        data = request.json
        emg_values = data.get("emg", [])

        if len(emg_values) != 10:
            return jsonify({"error": "10 EMG values required"}), 400

        input_data = np.array(emg_values, dtype=np.float32).reshape(-1, 10, 1)
        interpreter.set_tensor(input_details[0]['index'], input_data)
        interpreter.invoke()

        output = interpreter.get_tensor(output_details[0]['index'])
        prediction_index = int(np.argmax(output))
        prediction_label = class_labels[prediction_index]

        base_angle = int(emg_values[-1] * 100)
        base_angle = max(0, min(base_angle, 90))

        return jsonify({
            "prediction": prediction_label,
            "label": prediction_index,
            "angle": base_angle,
            "confidence": output[0].tolist()
        })

    except Exception as e:
        return jsonify({"error": str(e)}), 500

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Listing 4. Python Code to load TFLite model and and get prediction

```python
import serial
import requests
from collections import deque
import time

ser = serial.Serial('COM8', 115200, timeout=1)
server_url = "http://localhost:5000/predict"

emg_buffer = deque(maxlen=10)

try:
    while True:
        line = ser.readline().decode('utf-8').strip()
        if line and line.isdigit():

            normalized_emg = int(line) / 1023.0
            emg_buffer.append(normalized_emg)

            if len(emg_buffer) == 10:
                print("\nBuffer full. Sending to server...")
                try:

                    response = requests.post(server_url, json={"emg": list(emg_buffer)})

                    if response.status_code == 200:
                        data = response.json()
                        label = data.get("label", -1)
                        angle = data.get("angle", 0)

                        if label != -1:
                            command = f"{label}:{angle}\n"
                            ser.write(command.encode())
                            print(f"Sent to Arduino: {command.strip()}")
                        else:
                            print("Error: Invalid label from server")
                    else:
                        print(f"Server error: {response.text}")

                except Exception as e:
                    print(f"Request failed: {e}")

except KeyboardInterrupt:
    ser.close()
    print("\nProgram terminated.")
```

Listing 5. Python Code to send EMG to Flask, label and angle to Arduino

```cpp
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Servo.h>

#define EMG_PIN A0
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
#define SERVO_PIN 9

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
Servo therapyServo;

String inputString = "";
bool newData = false;

void setup() {
  Serial.begin(115200);
  therapyServo.attach(SERVO_PIN);
  therapyServo.write(90);

  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    while (1);
  }
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 10);
  display.println("Waiting for data...");
  display.display();
}

void loop() {
  static unsigned long prevEMGTime = 0;
  if (millis() - prevEMGTime >= 10) {
    prevEMGTime = millis();
    int emgValue = analogRead(EMG_PIN);
    Serial.println(emgValue);
  }

  while (Serial.available()) {
    char inChar = (char)Serial.read();
    if (inChar == '\n') {
      newData = true;
      break;
    } else {
      inputString += inChar;
    }
  }

  if (newData) {
    Serial.print("[DEBUG] Received: ");
    Serial.println(inputString);

    int separatorIndex = inputString.indexOf(':');
    if (separatorIndex != -1) {
      int label = inputString.substring(0,
          separatorIndex).toInt();
      int angle =
          inputString.substring(separatorIndex +
          1).toInt();
      updateDisplayAndServo(label, angle);
    } else {
      Serial.println("Invalid format!");
    }

    inputString = "";
    newData = false;
  }
}

void updateDisplayAndServo(int label, int
    baseAngle) {
  int therapyAngle = min(baseAngle + 5, 90);

  String severity;
  switch (label) {
    case 0: severity = "No CTS"; break;
    case 1: severity = "Mild CTS"; break;
    case 2: severity = "Moderate CTS"; break;
    case 3: severity = "Severe CTS"; break;
    default: severity = "Unknown"; break;
  }

  display.clearDisplay();
  display.setTextSize(2);
  display.setCursor(0, 20);
  display.print(severity);
  display.display();

  therapyServo.write(therapyAngle);
  Serial.print("[DEBUG] Moved servo to: ");
  Serial.println(therapyAngle);
}
```

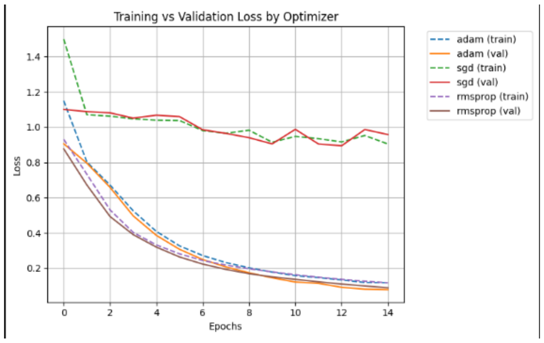
Listing 6. Arduino Code to read EMG, display on OLED and activate servo

TABLE II
COMPARATIVE ANALYSIS OF RNN CONFIGURATIONS FOR EMG CLASSIFICATION

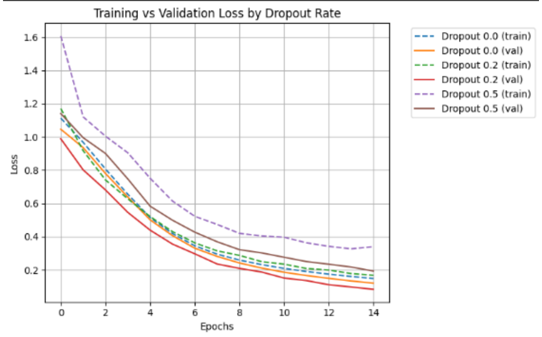
Configuration Low (0.4)	**Initial Loss** Steady heightReLU	**Final Loss** Very High (9.0)	**Convergence** Tanh Low (0.5)
Sigmoid	Medium (1.8)	Low (0.6)	Steady
		Optimizers	
Adam	Medium (0.9)	Very Low (0.1)	
SGD	High (1.5)	High (0.9)	Very slow
RMSprop	Medium (1.1)	Very Low (0.1)	Fast
		Dropout Rates	
0.0	Medium (1.0)	Low (0.15)	Fast
0.2	Medium (1.1)	Low (0.2)	Medium
0.5	High (1.2)	Medium (0.3)	Slow
		Number of Units	
16	High (1.2)	Medium (0.25)	Slow
32	Medium (1.0)	Low (0.15)	Medium
64	Low (0.7)	Very Low (0.05)	Fast

APPENDIX

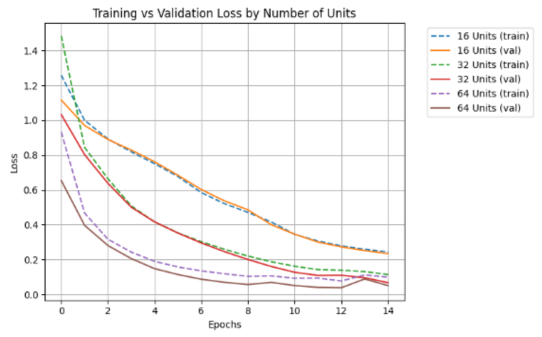
APPENDIX B: ADDITIONAL FIGURES



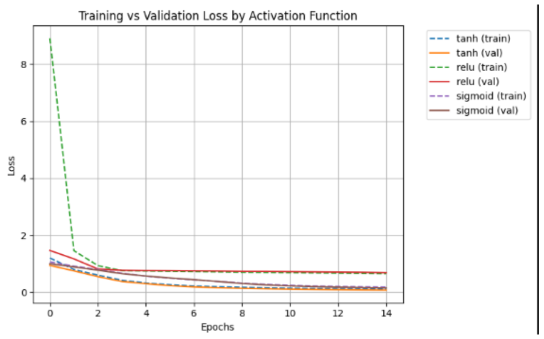
Training Vs Validation Loss by Optimizer



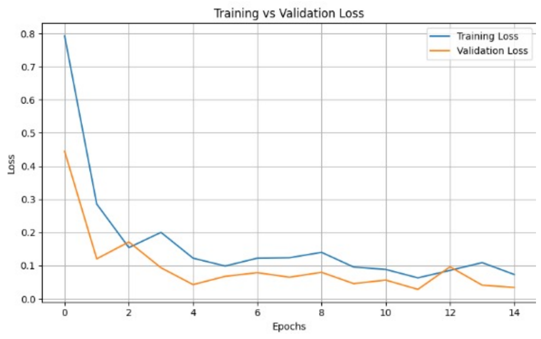
Training vs Validation Loss by Dropout rate



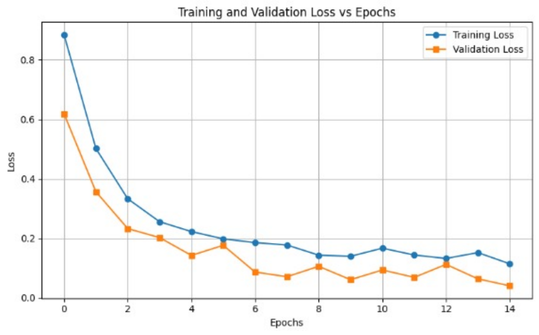
Training vs validation loss by number of units



Train vs validation loss by Activation Function



textbf Training Vs Validation Loss LSTM



Training Vs Validation Loss RNN