# Chapter 5

# Leveraging Deep feature learning for sEMG-based dynamic hand gesture recognition

This chapter focuses on recognizing dynamic hand gestures using sEMG, particularly handwritten character gestures. Section 5.1 introduces the problem, provides a formal description, and offers an overview of the solution. The experimental setup is described in Section 5.2, which details the materials and methods used. Section 5.3 presents preliminary concepts and outlines the proposed framework. Section 5.4 presents the experimental results and analysis achieved with the proposed technique. Finally, Section 5.5 concludes with a summary and outlines the future scope of this chapter.

## 5.1 Introduction

Despite rapid technological advancements, handwritten characters still hold significant roles in various fields, including education [31–33], communication, biometric signature verification [26, 27], and health care [34, 35, 78]. These applications often

require digitization of the handwritten characters and associated hand movements to facilitate effective analysis and interpretation of the underlying task. Offline and online handwriting recognition are crucial steps involving the digitization of handwritten characters [75]. Most existing systems extensively utilize image-processing techniques, which are highly sensitive to environmental lighting conditions. In contrast, surface electromyography (sEMG) signals, being invariant to lighting conditions, are employed in online handwriting recognition to facilitate the automatic transcription of handwritten characters.

Despite the advantages offered by sEMG signals, there has been limited effort to evaluate and analyze their efficacy in modeling handwriting movements [14]. Additionally, sEMG susceptibility to noise remains a significant challenge. While a few studies have addressed the decoding of handwritten words using sEMG signals, their efforts can be broadly categorized into two groups. First, propose efficient mathematical models to reconstruct handwritten traces. Secondly, others aim to develop efficient recognition models for identifying handwritten characters [81].

In this chapter, we utilize deep feature representation learning [36] to construct an effective and robust sEMG-based Handwritten Character Recognition (HCR) pipeline. Representation learning is the process of learning data representations that facilitate the extraction of useful information for the construction of classifiers or other predictive models [254]. We employ a stacked sparse denoising autoencoder network to extract efficient deep features, which are then integrated into traditional classifiers for the HCR task. For experimentation, we collect new datasets comprising sEMG signals corresponding to 26 handwritten lowercase English alphabets from 15 subjects. Rigorous evaluations are conducted to validate the results. Based on the results obtained, Our proposed pipeline holds potential for real-time Human-Computer Interaction (HCI) applications in smart classrooms, facilitating the digitization of handwritten notes and in clinical applications [35, 129, 255] involving handwriting analysis tasks.

### 5.1.1 Problem Statement and Overview of the Solution

Handwritten character recognition using sEMG signals presents significant challenges due to the dynamic and variable nature of handwriting movements and the presence of signal noise. Traditional image-based recognition systems, such as optical character recognition [256], are less efficient in capturing the underlying muscle activities involved in handwriting. This chapter aims to develop a robust and efficient Handwritten Character Recognition (HCR) system that leverages the potential of muscle activities for improved accuracy.

**Definition (Deep Feature Representation Learning)** Deep feature representation learning involves utilizing deep neural networks to automatically discover and extract meaningful, high-level features from raw input data. Formally, let $\mathbf{x}$ be the input data. The deep neural network applies a series of transformations, such that the activations $\mathbf{h}$ at layer $l$ are given by $\mathbf{h}^{(l)} = \sigma(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)})$, where $\mathbf{h}^{(0)} = \mathbf{x}$. Here, $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the weights and biases of the $l$-th layer, respectively, and $\sigma$ is the activation function, such as ReLU or sigmoid. The final layer $\mathbf{h}^{(L)}$ represents the **deep features**, capturing the underlying patterns and structures in the input data. These features can be used for various machine-learning tasks. The deep network is trained to minimize a loss function $\mathcal{L}$, such as the mean squared error (MSE) for regression tasks or cross-entropy for classification tasks.

#### 5.1.1.1 Problem Statement

Given a dataset $D$ consisting of $N$ sEMG time series recordings corresponding to 26 English alphabets, each recording $TS_i(u) = \{u_1, u_2, u_3, \ldots, u_N\}$ for $i$ in the range 1 to 8, represents sEMG signals collected from eight sensors. How can we accurately predict the class label of each handwritten character using deep features extracted from these sEMG signals? The primary challenge is to leverage deep representation learning techniques to extract high-level features from sEMG signals.

Furthermore, sEMG signals are susceptible to noise, which can significantly impact the accuracy of the recognition model. To this end, we aim to investigate the feasibility of using denoising autoencoders (DAEs) to develop a noise-robust sEMG recognition model.

#### 5.1.1.2 Overview of the Solution

To meet the objective, the subsequent tasks are arranged as a pipeline:

**Data Acquisition:**Collect sEMG signals corresponding to the 26 English alphabets from multiple subjects using the Myo armband. The resulting dataset, denoted as $S - HCR_{\text{raw}}$, can be represented as:

$$S - HCR_{\text{raw}} = \{TS_1(u), TS_2(u), TS_3(u), TS_4(u), \ldots, TS_q(u)\}$$

where $TS_i(u)$ is a time series with $N$ data points from a single sensor.

**Preprocessing**: Apply digital filters to the raw sEMG signals to remove noise and enhance signal quality.

**Feature Extraction**: Extract various time and frequency domain features for a fixed window size $W_s$. The features extracted for each window are combined into a single file for each alphabet, creating 26 classes corresponding to the English alphabet. The labeled feature set for a particular class $C_k$ is defined as:

$$C_k = \{f_{i,1}, f_{i,2}, f_{i,3}, \ldots, f_{i,m}, l_k\}$$

where $1 \leq k \leq 26$, $f_{i,m}$ represents a feature extracted for window size $W_s$, and $l_k$ is the label assigned to class $C_k$.

**Deep Feature Representation Learning**: Use a modified deep autoencoder architecture to transform the extracted features into deep features, capturing complex and hierarchical information about handwriting movements.

**Classification**: Integrate the deep features into conventional classifiers, such as Support Vector Machines (SVM), to predict the class labels of the handwritten characters.

**Experimental Evaluation**: Conduct extensive experiments to assess the performance of the proposed HCR system and validate the results using standard performance metrics. Figure 5.1 summarized the generalized pipeline proposed for the handwritten character recognition task.

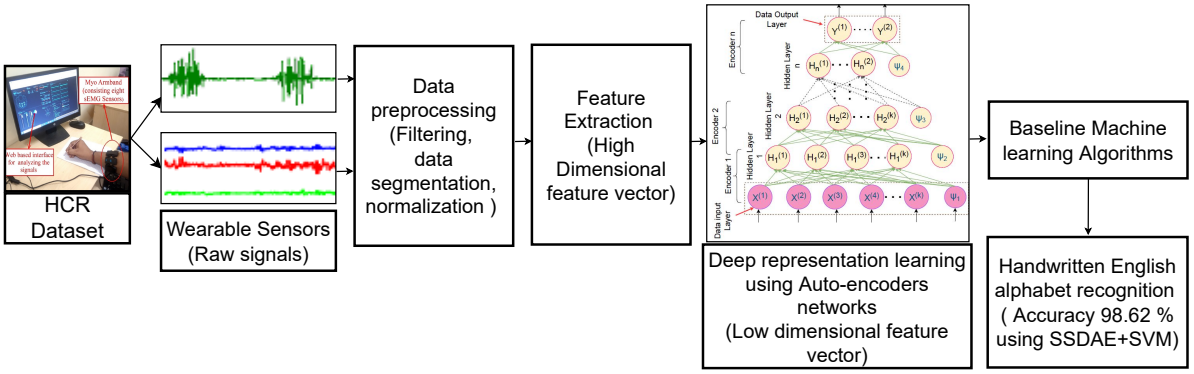



FIGURE 5.1: Generalized pipeline proposed for Handwritten character recognition task

The proposed approach aims to address the challenges of handwriting recognition using sEMG signals, offering a robust and accurate solution for various practical applications. To validate the effectiveness of the proposed model, we formulated the following research questions(RQs):

1. RQ8: *How effective are proposed stacked sparse denoising autoencoders in extracting meaningful features for dynamic hand gesture recognition using sEMG signals? ( Addressed in Section 5.4.2)*

2. RQ9: *How does the proposed sEMG-based recognition model perform in scenarios with varying noise levels? (Addressed in Section 5.4.2.1)*

3. RQ10: *How do stacked sparse denoising autoencoders enhance feature extraction for dynamic hand gesture recognition using sEMG signals compared to traditional denoising autoencoders? ( Addressed in Section 5.4.2.2)*

4. RQ11: *How does the proposed deep learning pipeline perform compared with existing state-of-the-art methods for sEMG-based dynamic hand gesture recognition in terms of accuracy, robustness, and computational efficiency? ( Addressed in Section 5.4.4)*

### 5.1.2 Major contribution of the work

Significant contributions of the chapter are as follows

a) We proposed an efficient wearable sensor-based Handwritten Character Recognition (HCR) pipeline that can easily differentiate between complex hand movements and correctly classify different handwritten characters (26 lowercase English alphabets).

b) A stacked sparse denoising autoencoder(SSDAE) is used to extract and compress features effectively. SSDAE helps to obtain a more representative feature set from the original features, enhancing the classification process(HCR).

c)The effectiveness and robustness of the proposed pipeline have been verified through standard performance metrics, achieving recognition accuracy as high as 98.72% even with the traditional machine learning classifiers.

d)We collected a new dataset from 15 subjects, consisting of sEMG signals corresponding to 26 English alphabets written using pen and paper (S-HCR dataset).

## 5.2 Materials and methods

### 5.2.1 Sensor: Surface electromyography(sEMG)

In this study, we mainly focus on Surface Electromyography (sEMG) sensors to gather comprehensive data for hand gesture recognition. The sEMG is a noninvasive technique that records and detects electrical signals generated by muscles during

motion or at rest. These sensors are used to collect raw signals for 26 English lowercase alphabets (Details of sEMG provided in section 1.0.2)

#### 5.2.1.1 Data collection device

Myo armband, a cost-effective wearable sensor developed by Thalamic Lab, was used to collect the sensory data [164]. A comprehensive description of the Myo armband and its sensors is provided in Section 2.3.2.1.

### 5.2.2 Number of subjects and experimental protocol

Fifteen healthy people (eleven males and four females) aged 28 -32 years participated in the data collection. The dataset was collected in the scenario where each subject was made to write 26 lowercase English alphabets using a pen on paper. A wearable sensor with a wireless eight-channel sEMG was applied to collect 26 lowercase English alphabet signals. The device has a sampling frequency of 200Hz. The raw signals corresponding to different sensors from each subject were stored and processed to form different datasets.

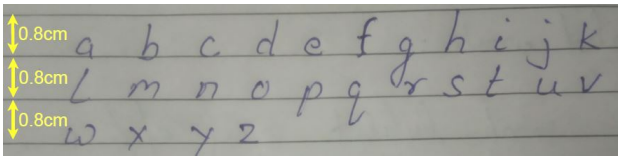



FIGURE 5.2: The figure illustrating the template corresponding to 26 lower case English alphabet used for collected raw signals

The dataset collection was performed in separate sessions, collecting different samples at a time. A gap of a few seconds is maintained between two consecutive sessions to reduce the effect of muscle fatigue. Before placing the sensors, the subject's hair was removed from the skin's target portion, and the skin was cleaned with alcohol. The sensors were placed on the forearm near the wrist region, as the muscles are directly responsible for the movement of the wrist and fingers. Each of the eight sensor blocks was numbered from 1-8, and an effort was made to place the sensors in a similar position on different subjects.

While writing characters, each of them was written between two parallel lines, 0.8 cm apart, and running throughout the width of the page. This approach is a common practice used for early learners and, more specifically, restricts each character's maximum font size. Moreover, a fixed template displaying the lowercase English alphabet was used as a reference case for all subjects. Figure. 5.2 shows the template for lowercase English alphabets used for pen and paper-based modality. Figure. 5.3 illustrates the experiment setup including the placement of sensors (Fig. 5.3 a) and data collection scenario (Fig. 5.3 b) used during data collection. In contrast, (Fig. 5.3 c) shows the experimental setup used by deploying the trained model on a low-cost device (Raspberry Pi). We have used the web-based measurement setup provided by the Thalmic Labs to monitor incoming signals during data collection. The measurement setup can be accessed through url[1].

### 5.2.3 Dataset preparation (S-HCR)

The S-HCR dataset comprises sEMG signals collected for the 26 English alphabets from multiple subjects. The data points generated over a fixed duration can be represented as a separate time series for each sensor.

Let $TS_i(u) = \{u_1, u_2, u_3, \ldots, u_N\}$ be a time series with $N$ data points corresponding to sEMG signals collected from a single sensor. For eight sensors, the resulting collection of raw signals can be depicted as a multivariate time series given by:

$$S - HCR_{\text{raw}} = \{TS_1(u), TS_2(u), TS_3(u), TS_4(u), \ldots, TS_q(u)\}$$

Various time and frequency domain features were extracted for a fixed window size $W_s$. For each alphabet, features extracted for these windows were combined into a single file. Twenty-six classes corresponding to the 26 English alphabets were created for the recognition task. Each class, $C_k$, was assigned a distinctive

[1]https://diagnostics.myo.com

label $l_k$, $1 \leq l_k \leq 26$, to facilitate supervised classification. The labeled feature set corresponding to a particular class $C_k$ can be defined as:

$$C_k = \{f_{i,1}, f_{i,2}, f_{i,3}, \ldots, f_{i,m}, l_k\}$$

where $C_k$, $1 \leq k \leq 26$, is the class corresponding to each alphabet. $f_{i,m}$ represents a feature extracted for a window size $W_s$, where $i$ is the particular feature instance and $m$ is the total number of features extracted. A detailed description of the dataset is provided in Section 2.3.8.

#### 5.2.3.1 Filters

The EMG signal typically lies within the frequency range of 0-500 Hz. However, for practical purposes, the most dominant frequency is usually considered from a range slightly above 0 Hz [257]. In our implementation, we utilized the range of 5-500 Hz. To address the noise encountered during data acquisition, a third-order digital Butterworth band-pass filter with a bandwidth of 5-500 Hz was applied to the sEMG signals [73,246]. Additionally, a configurable Butterworth filter was employed to eliminate the 50 Hz power line interference. [246]

#### 5.2.3.2 Data segmentation

Data segmentation is employed to identify the appropriate segment of the raw signals that contains sufficient information to accurately classify each handwritten character. During our experiment, we observed that the time required to write an individual English alphabet varies depending on the character's font shape. For instance, the time taken to write the character 'c' is less than that for 'k' across all subjects. To accommodate this variation, we selected the average time duration needed to write a single alphabet as the segment length (window size).

Due to the complexity of measuring the exact time taken to write a single character, we instead counted the frequency of each alphabet written over a 10-second

interval by various subjects. To ensure generalization, we calculated the average writing frequency across all subjects. For most English alphabets, the frequency ranged from 14 to 20 per 10 seconds. Figure 5.4 illustrates the writing frequency of each alphabet over a 10-second period.

To more accurately estimate the average writing frequency of an alphabet, we calculated a weighted average of these frequencies rather than using a simple mean. The weights were based on the total number of alphabets corresponding to each specific frequency. For example, eight out of 26 alphabets were typically written 16 times in ten seconds on average. Thus, the frequency '16' was assigned a weight of $\frac{8}{26} = 0.3076$.

This weighted average provided a generalized writing frequency (per 10 seconds) for the alphabet and was used to calculate the average time duration needed to write a single alphabet. The average time duration was found to be approximately 0.6 seconds. Using this 'average time' and the sampling frequency of the sensors, we determined the window length. Given that the sEMG sensors operate at 200Hz, the window size $W_s$ for the S-HCR dataset was set to 120 samples (0.6 x 200).

### 5.2.4 Feature extraction

The quality of features extracted from raw signals significantly impacts the performance of control schemes used in myoelectric and prosthetic devices [217] [258] [259]. One of the significant contributions of our chapter is finding a suitable representative feature set for the sEMG sensor-based handwriting character classification task. In order to cover a broader range of relevant features, we used the significant features provided in the literature for sEMG-based handwriting recognition and other related tasks [217] [10] [260]. Efforts were made to include most of the features used for upper-limb prosthetic control [217] as the muscles responsible for upper limb movements are generally used during handwriting movements. The extracted features mainly contain time, frequency, and time-frequency domain features.

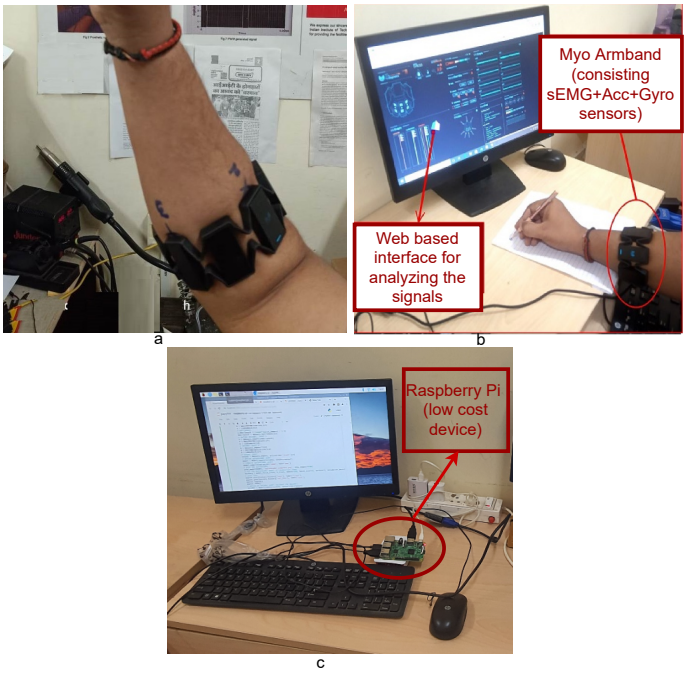



FIGURE 5.3: The experimental set-up used for data collection

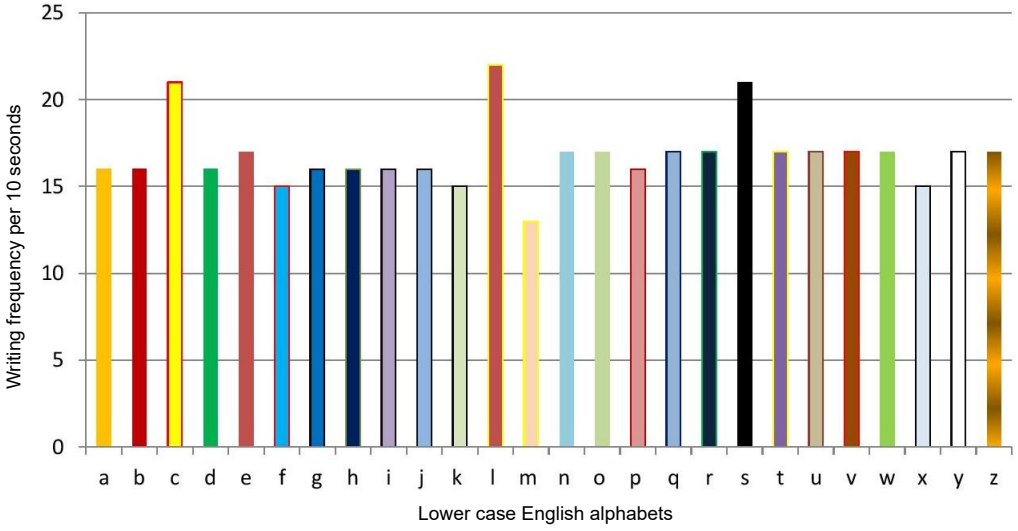



FIGURE 5.4: Alphabets writing frequency used to calculate the time taken to write a single character

Some of the important features extracted include Entropy, Benford Correlation, c3 non-linearity statistics [220], Complexity-invariant distance (complexity information) [221], Mexican hat wavelet [222], Spectral centroid (absolute), skewness, kurtosis of the Fourier transform, Power spectral density based on Welch Method [223], Lempel-Ziv based complexity [224], One-dimensional Matrix profile [225], Partial

autocorrelation, Root Mean square, Sample entropy, Friedrich coefficients [226], Absolute sum of changes (consecutive time series value changes), Langevin fixed point (Largest point of deterministic dynamics) and Quantiles. Moreover, statistical features such as Kurtosis, Mean, Variance, Absolute energy, Autocorrelation, and Standard deviation were also calculated. Time domain signals were transformed into the frequency domain using the Fast Fourier Transform (FFT). As various hand movements mainly produce lower frequency distributions, we choose the first 100 FFT coefficients as features [67, 218, 219]. For implementation, source code provided by the open-source library [261] was used.

## 5.3 Preliminary Concepts and Proposed Architecture

As discussed earlier, this chapter proposes a framework for sEMG-based Handwritten Character Recognition (HCR) that utilizes deep representation learning techniques. The core component of this framework is a Stacked Sparse Denoising Autoencoder (SSDAE) network, which is employed for deep feature extraction. The SSDAE combines the benefits of sparse and denoising autoencoders, enabling it to learn compact and robust feature representations by reconstructing the original input from corrupted versions, imposing sparsity constraints on the hidden layers. The deep features extracted by the SSDAE were integrated into conventional classifiers, such as Support Vector Machines (SVM), to achieve improved accuracy in recognizing handwritten characters. This section systematically introduces the preliminaries and subsequently details our proposed deep-stacked denoising autoencoder. Section 5.3.1 introduces the autoencoder. Meanwhile, Section 5.3.2 presents the concept of Stacked Autoencoders, followed by the introduction of sparsity and denoising concepts in Sections 5.3.3 and 5.3.4, respectively. Finally, the Stacked Sparse Denoising Autoencoder (SSDAE) is described in detail, outlining its architecture, training process, and integration into the overall objective function.

### 5.3.1 Autoencoder

An autoencoder is a neural network (unsupervised algorithm) that tries to learn an approximation to an identity function that helps to reconstruct the given input equal to target values. The network consists of an input, hidden, and output layer. Based on the functionality, the architecture can be grouped into two modules: an encoder and a decoder. Initially, using the encoder module, the autoencoder tries to transform a higher dimensional input feature vector into low dimensional feature space. The encoder module processes the input to the hidden layers, assuming that,

$$H = \mu_1(I) = \sigma(W^{(1)}I + \Psi^{(1)}) \tag{5.1}$$

The output of the autoencoder is achieved using the decoding module,

$$J = \mu_2(H) = \sigma(W^{(2)}H + \Psi^{(2)}) \tag{5.2}$$

The decoder module reconstructs the input features from the low-level representations produced by the encoder module at the hidden layer. Figure. 5.5 illustrates the autoencoder having a single hidden layer.

In the above equations $I \in \mathbb{R}^n$, is the input vector, $\mu_i(*)$ is a non linear activation function, $H \in R^{n'}$, is the lower dimension vector output from the $n'$ hidden layers. $J \in \mathbb{R}^n$, is the output feature vector reconstructed using the autoencoder. $W^{(1)} \in R^{n'*n}$, is the weight matrix for the connection between the input layer and the network hidden layer, while the $W^{(2)} \in R^{n*n'}$is the weight matrix associated with hidden layer and the output layer.$\Psi^{(1)} \in \mathbb{R}^n$ and $\Psi^{(2)} \in \mathbb{R}^{n'}$ are the bias vectors corresponding to input and the hidden layers, respectively. For the input dataset consisting of k examples, the neural network is trained to minimize the reconstruction error $\frac{1}{k}\sum_{i=1}^{k}(J_i - I_i)^2$ by applying the backpropagation algorithm.

By adjusting the parameters $W^{(1)}$,$W^{(2)}$, $\Psi^{(1)}$ and $\Psi^{(2)}$, an effort is made to minimize the error between the original and the reconstructed input. The overall cost function ($J_{cost}(W, \Psi)$) is defined as:

$$J_{cost}(W, \Psi) = \frac{1}{k} \sum \frac{1}{2} \|J - I\|^2 + \frac{\lambda}{2} \sum (W)^2$$

The initial term of the cost function is mean square error, while the second term is referred to as the regularization term.The regularization term is used to deal with the overfitting problem. $z$ is the number of layers, and $s_u$ is the number of hidden layer units. Initially, the parameters $W$ and $\Psi$ are assigned random values close to zero, but during model learning, eventually, their optimal values are obtained using gradient descent.

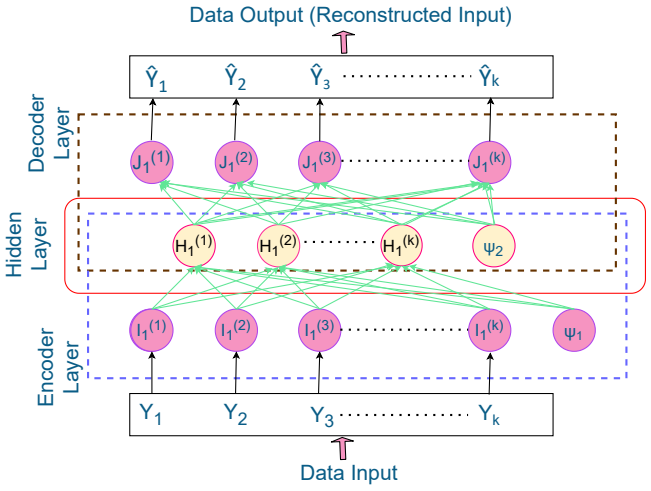



FIGURE 5.5: A single hidden layer autoencoder architecture

### 5.3.2 Stacked Autoencoder

A stacked autoencoder (SAE) is a modified auto-encoder neural network in which multiple autoencoders are arranged sequentially. A simple autoencoder might produce representations similar to those from Principal Component Analysis (PCA) [262] and struggle to capture complex nonlinear features due to its single hidden

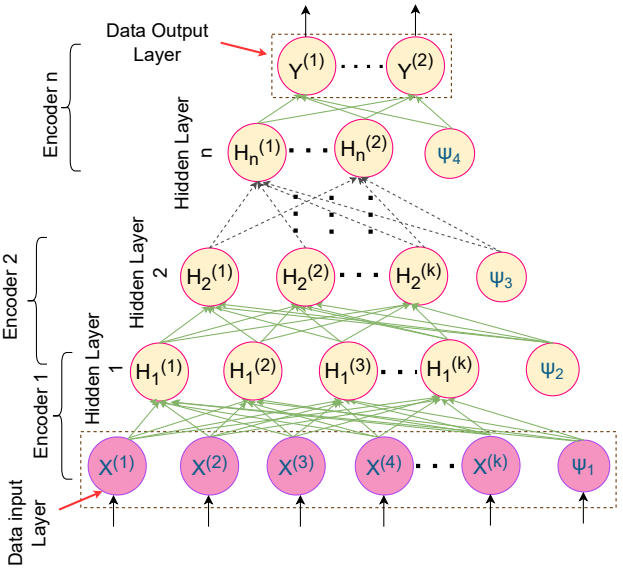



FIGURE 5.6: Stacked autoencoder structure

layer. This limitation persists even with nonlinear activation functions. To address this, additional hidden layers are introduced to learn intricate structures and patterns within the data effectively.

Stacked autoencoders can uncover more complex, high-level features by increasing the number of layers and employing appropriate activation functions. This architecture enables the learning of more representative and comprehensive features compared to a single hidden layer autoencoder [36, 254]. Essentially, a stacked autoencoder with $m$ hidden layers can be viewed as a combination of $m$ individual autoencoders trained in a greedy layer-wise manner. Figure 5.6 illustrates a stacked autoencoder with $n$ encoder units connected in sequence.

### 5.3.3 Stacked Sparse Auto-encoder (SSAE)

One of the limitations of a simple auto-encoder is that it may learn a trivial identity function for the training dataset. Such a function can successfully reconstruct the original input at the output layers without learning any meaningful feature representation. So, the efficiency of the neural network is not generalized for the new inputs.

To deal with this issue, the neural network can be made to learn interesting structures in the input data by putting constraints on the hidden layers (bottleneck layer). For example, reducing the number of units in hidden layers can force the network to learn compressed feature representation. Due to the reduced number of units, the network tries to retain the most significant features in the hidden layers. The decoder module further utilizes these reduced features to reconstruct the original input.

However, without significantly reducing the number of units at hidden layers, the network can still be forced to learn an effective and reduced feature representation. This can be achieved by adding a sparsity constraint at the hidden layers [262]. The sparsity constraint ensures that the average activation at each node is close to zero. For such a network, it is assumed that neurons are suppressed most of the time, hence named sparse constraint.

For the network, let the average activation for a single unit, given k input is $\hat{\rho}_t$, which is defined as

$$\hat{\rho}_t = \frac{1}{k}\sum_{i=1}^{k}[\alpha_t^{(l)}(\hat{x}^{(i)})] \tag{5.3}$$

Here, the $\alpha_t^{(l)}$ is the activation value for the $t^{th}$ hidden unit of the autoencoder network. Whereas, $\alpha_t^{(l)}(\hat{x}^{(i)})$ be the activation value of the very hidden unit obtained due to the input $\hat{x}^{(i)}$. To apply the sparsity constraint, the average activation ($\hat{\rho}_t$) is made equivalent to a constant $\rho$. $\rho$ is the sparsity parameter usually taken close to zero. For the implementation, an extra penalty is added to the overall cost function, penalizing $\hat{\rho}_t$ when it significantly deviates from the assigned sparsity parameter. This additional penalty is based on Kullback-Leibler(KL) divergence and is defined as:

$$KL_{div}(\rho \parallel \hat{\rho}_t) = \sum_{m=1}^{S}(\rho \log \frac{\rho}{\hat{\rho}_t} + (1-\rho)\log \frac{1-\rho}{1-\hat{\rho}_t}) \tag{5.4}$$

KL divergence is a measure to calculate the difference between two distributions.The added penalty has the property $KL_{div}(\rho \parallel \hat{\rho}_t)$=0, when the $\rho = \hat{\rho}_t$ and its value tends to infinity as the $\hat{\rho}_t$ approaches zero or one. To accommodate this penalty, the overall Cost function is modified as

$$J_{\text{spr\_cost}}(W, \psi) = J_{\text{cost}}(W, \psi) + \beta \sum_{m=1}^{S} KL_{\text{div}}(\rho \parallel \hat{\rho}_t) \tag{5.5}$$

$$J_{\text{spr\_cost}}(W, \Psi) = \frac{1}{k} \sum_{i=1}^{k} \frac{1}{2} \left\| J_i - I_i \right\|^2 + \frac{\lambda}{2} \sum_{l=1}^{z-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (W_{ij}^{(l)})^2 + \beta \sum_{m=1}^{S} KL_{\text{div}}(\rho \parallel \hat{\rho}_t) \tag{5.6}$$

Where $J_{cost}(W, \Psi)$ is the Cost function, $\beta$ is the constant that controls the weight of the sparse penalty term. For our experiment, we take $\beta$ values near to 0.01.

### 5.3.4 Denoising auto-encoder(DAE)

The factors such as model complexity and training dataset size may cause an over-fitting problem in the encoder network. To deal with the scenario, the autoencoder neural network is often trained using added noise, which eventually helps to increase the model's generalization ability.

For a denoising autoencoder (DAE), the input data is initially corrupted using random noise. This corrupted data is then forwarded to the encoder-decoder module. The added noise is the matrix with the same dimension as input data $I'$, while the degree of corruption is controlled by a factor $\eta$. The corruption operation (adding noise) can be defined as

$$\hat{I} = Rand(\eta) + I' \tag{5.7}$$

$\hat{I}$ is given as input to the autoencoder for further processing. Hence, equation get updated as:

$$\hat{H} = \mu_3(\hat{I}) = \sigma(W_{ih}\hat{I} + \Psi^{(1)}) \tag{5.8}$$

The output of the autoencoder is achieved using the decoding module,

$$\hat{J} = \mu_4(\hat{H}) = \sigma(W_{hj}\hat{H} + \Psi^{(2)}) \tag{5.9}$$

By giving the corrupted input, DAE, the model is forced to learn an intermediate representation at the hidden layer, which can be used by the decoder module to reconstruct the original non-corrupted input. The DAE tries to minimize the overall cost function ($J_{de\_cost}(W, \psi)$), which is expressed as follows:

$$J_{de\_cost}(W, \Psi) = \frac{1}{k}\sum_{i=1}^{k}\frac{1}{2}\left\|\hat{J}_i - \hat{I}_i\right\|^2 + \frac{\lambda}{2}\sum_{l=1}^{z-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ij}^{(l)})^2 \tag{5.10}$$

### 5.3.5 Proposed Stacked Sparse Denoising Autoencoder(SSDAE)

The proposed architecture is inspired by key observations from prior research. Bengio et al. [263] introduced stacked autoencoders and demonstrated that the greedy layer-wise unsupervised training strategy enhances optimization by initializing weights near a good local minimum. This method results in internal distributed representations that are high-level abstractions of the inputs, leading to better generalization. Vincent et al. [264] proposed stacked denoising autoencoders, which are trained to denoise corrupted versions of the inputs, thereby improving robustness and feature learning. These insights underpin the design of the Stacked Sparse Denoising Autoencoder (SSDAE).The functionality of the SSDAE network is achieved by providing corrupted input to the SSAE network. So, using equation 5.6 and equation 5.10, the overall objective function for the SSDAE is modified as;

$$J_{dss\_cost}(W, \Psi) = \frac{1}{k}\sum_{i=1}^{k}\frac{1}{2}\left\|\hat{J}_i - \hat{I}_i\right\|^2 + \frac{\lambda}{2}\sum_{l=1}^{z-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ij}^{(l)})^2 + \beta\sum_{m=1}^{S}KL_{div}(\rho \parallel \hat{\rho}_t)$$

Initially, the noise is added to the input data, performing corruption operations for the training process. Utilizing the modified cost function, the network tries to optimize the parameters(W,Ψ) using backpropagation for given set of $\rho$, $\beta$, and corruption rate $\eta$.

## 5.4 Results and Discussion

The effectiveness of the proposed pipeline was assessed through a series of experiments conducted on the collected dataset. The primary objectives of these experiments were as follows:
a) To evaluate and analyze the efficiency of the stacked sparse denoising autoencoder (SSDAE) for the handwritten character recognition (HCR) task.
b) To compare the performance of the proposed pipeline with other state-of-the-art methods.

The handwritten character recognition problem was framed as a multi-class classification task. Statistical metrics such as Accuracy, Precision (P), Matthews Correlation Coefficient (MCC), Recall (R), and F1 Score (F1) were utilized to evaluate the performance of the experiments.

### 5.4.1 SSDAE-based handwritten English character recognition using S-HCR dataset

The proposed approach, SSDAE, leverages the strengths of two robust autoencoders, the Stacked Sparse Autoencoder (SSAE) and the Denoising Autoencoder (DAE), to construct a hybrid model called the Stacked Sparse Denoising Autoencoder. This model is designed to identify the representative feature subset for the classification task. In the DAE network, random noise is introduced to the input, corrupting the original data. The model then attempts to reconstruct the input

from the noisy data, ultimately learning useful feature representations. This data corruption increases the model's robustness and ability to handle noise.

Conversely, SSAE employs a sparsity constraint by incorporating the KL Divergence function to penalize the objective function during training. The sparsity parameters limit the activation of hidden units, resulting in most units being inactive. This encourages the neural network to learn a compressed and meaningful representation in the bottleneck layer, ensuring that the network can accurately regenerate the original input at the decoder stage. This process produces representative features that are critical for the model's performance.

SSDAE integrates the benefits of both DAE and SSAE into a single network. Key structural parameters, such as the corruption rate, sparsity parameter, number of hidden layers, and the number of units in each hidden layer, were carefully defined for our Handwritten Character Recognition (HCR) task to optimize performance.

#### 5.4.1.1 Designing the SSDAE Network Architecture

In designing the proposed SSDAE neural network, the selection of parameters was guided by the need to accommodate the sparsity constraint. The Adam adaptive optimizer was chosen over the traditional gradient descent algorithm due to its adaptive behavior, which minimizes the need for learning rate tuning. While gradient descent updates network parameter weights uniformly, Adam dynamically adjusts the weights based on the first and second-order moments of the gradients. This approach, combined with the backpropagation algorithm, allows each layer of the stacked SSDAE network to achieve optimized weights and biases.

As for the deep neural network, there is no fixed method to find the appropriate structural parameters; we explored various configurations to maximize classification accuracy. The sparsity constraint value was initialized at 0.05, inspired by the work of Bengio et al. [265], which demonstrated that this value yields satisfactory experimental outcomes. Subsequently, we determined the number of hidden layers and the

number of hidden units per layer. A three-layer SSDAE was designed, with the first two hidden layers evaluated using 500, 400, 300, and 200 nodes. Following extensive testing of these configurations, we established that 100 nodes per hidden layer provided the best performance. Additionally, we varied the number of hidden layers within the SSDAE network. After considering the trade-off between computational complexity and recognition accuracy, we concluded that a three-layer architecture was optimal. Table 5.1 summarizes the SSDAE network parameters.

| **Parameter** | **Value** |
|---|---|
| Input Dimensions | 200 |
| Number of Layers | 6 |
| Activation Function | LeakyReLU |
| Regularization | SparseRegularizer |
| Regularization Parameters | rho = 0.05, beta = 0.01 |
| Noise Factor | [0.03- 0.9] |
| Optimizer | Adam |
| Loss Function | Mean Squared Error (MSE) |
| Epochs | 20 |
| Batch Size | 16 |
| Encoder Layer 1 | 400 neurons with BatchNormalization and LeakyReLU activation |
| Encoder Layer 2 | 200 neurons with BatchNormalization and LeakyReLU activation |
| Bottleneck Layer | 100 neurons |
| Decoder Layer 1 | 200 neurons with BatchNormalization and LeakyReLU activation |
| Decoder Layer 2 | 400 neurons with BatchNormalization and LeakyReLU activation |

TABLE 5.1: Proposed SSDAE parameters and Layer Details

### 5.4.2 Performance Measures

We have employed standard evaluation metrics, including average accuracy, MCC score, ROC-AUC curve, and kappa as performance measures. A detailed discussion of these metrics is provided in Section 2.3.12. These metrics offer a comprehensive assessment of the model's performance across various dimensions.

These evaluation metrics were primarily applied to various conventional machine learning classifiers to validate the effectiveness and generalizability of the learned SSDAE features by comparing their classification results. Table 5.2 compares the classification accuracy of different classifiers—Linear Discriminant Analysis (LDA), K-Nearest Neighbors (K-NN) with K=3 and K=5, and Support Vector Machine (SVM)—to evaluate the efficiency and discriminatory power of the learned representation.

LDA demonstrates consistent accuracy, ranging from 96.03% to 96.39%, indicating its reliability and stability in classification tasks. LDA is a standard classifier and frequently used in sEMG-based classification due to its less complex nature and faster processing speed [230, 231]. K-NN, with K=3 and K=5, achieves mean classification accuracy between 93.58% and 94.90%.

SVM exhibits the highest accuracy among the classifiers, ranging from 98.10% to 98.62%, highlighting its superior performance and robustness. The effectiveness of the learned representation is evidenced by the improved results achieved even by baseline classifiers, demonstrating its discriminatory ability.

Table 5.2 provides the mean Matthews correlation coefficient (MCC) obtained through ten-fold cross-validation. MCC is a robust statistical metric for classification performance, particularly effective for imbalanced datasets [183,266]. MCC exhibits a similar pattern to that suggested by accuracy metrics.

The combination of SSDAE and SVM was thoroughly validated using additional performance metrics, such as Precision, Recall, AUC-ROC curve, and F1 score, to confirm its effectiveness and robustness.

Figure 5.7 presents the ROC-AUC curve for the classification results related to handwritten character recognition. The ROC-AUC curve plots the true positive rate against the false positive rate and helps to visualize the trade-offs between sensitivity and specificity, providing a comprehensive view of the classifier's performance in distinguishing between different handwritten characters. Additionally, Table 5.3 highlights the Precision, Recall, and F1 scores for the 26 classes corresponding to different lowercase English alphabets, which are visualized in Figure 5.8. The model reports the lowest precision, recall, and F1 scores for the letters 's', 't', and 'u', indicating a struggle to predict these characters accurately. In contrast, the model efficiently recognizes the letters 'c', 'e', 'p', 'x', and 'y', as reflected in their higher precision, recall, and F1 scores.

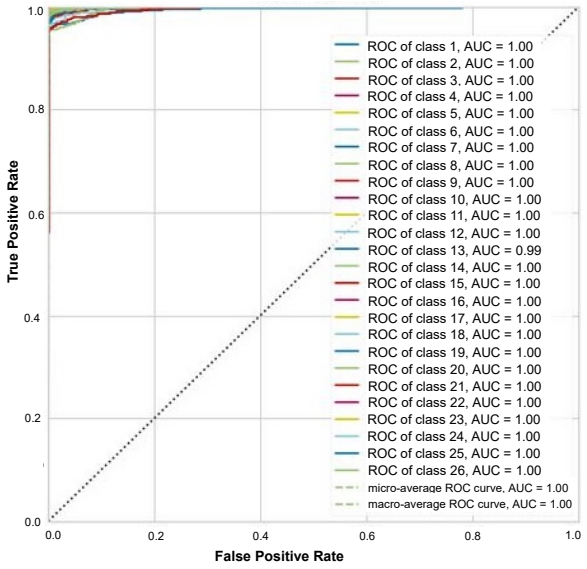
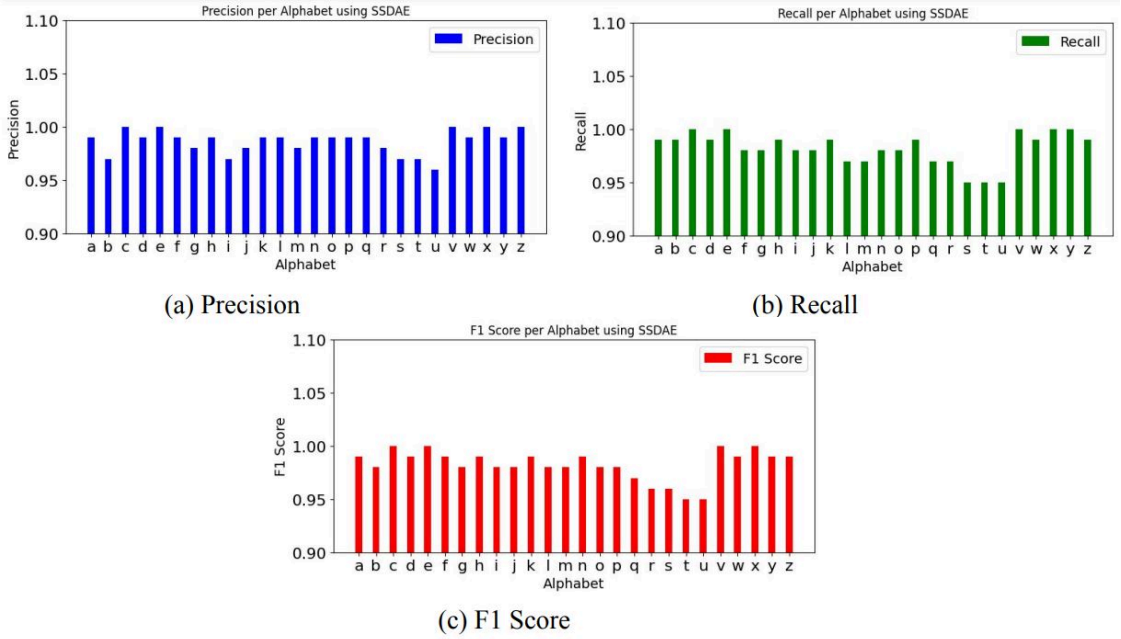



FIGURE 5.7: ROC AUC curve obtained for SSDAE + SVM on S-HCR dataset




(a) Precision (b) Recall

(c) F1 Score

FIGURE 5.8: Precision, Recall, and F1 Score obtained for SSDAE + SVM on S-HCR dataset

TABLE 5.2: The classification accuracy and mean MCC score achieved using SSDAE network and baseline algorithms

| | LDA | | K-NN(K=3) | | K-NN(K=5) | | SVM | |
|---|---|---|---|---|---|---|---|---|
| $\eta$ | Accuracy | MCC | Accuracy | MCC | Accuracy | MCC | Accuracy | MCC |
| 0.01 | 96.28±.16 | 0.9613 | 94.44±.16 | 0.9421 | 94.55±.14 | 0.9413 | 98.61±.06 | 0.9855 |
| 0.03 | 96.03±.12 | 0.9587 | 93.58±.11 | 0.9353 | 93.89±.19 | 0.9364 | 98.48±.10 | 0.9842 |
| 0.1 | 96.22±.11 | 0.9609 | 94.18±.15 | 0.9411 | 94.70±.13 | 0.9465 | 98.56±.05 | 0.9846 |
| 0.2 | 96.29±.13 | 0.9614 | 94.34±.11 | 0.9418 | 94.40±.16 | 0.9417 | 98.53±.15 | 0.9850 |
| 0.3 | 96.39±.16 | 0.9624 | 94.46±.14 | 0.9423 | 94.47±.14 | 0.9425 | 98.50±.08 | 0.9844 |
| 0.4 | 96.28±.17 | 0.9613 | 94.66±.13 | 0.9445 | 94.90±.14 | 0.9440 | 98.62±.11 | 0.9861 |
| 0.5 | 96.20±.13 | 0.9605 | 94.38±.14 | 0.9416 | 94.36±.14 | 0.9414 | 98.45±.08 | 0.9843 |
| 0.6 | 96.09±.12 | 0.9594 | 94.58±.11 | 0.9436 | 94.51±.11 | 0.9429 | 98.40±.12 | 0.9839 |
| 0.7 | 96.23±.14 | 0.9607 | 94.44±.13 | 0.9421 | 94.25±.14 | 0.9401 | 98.29±.06 | 0.9828 |
| 0.8 | 96.35±.13 | 0.9620 | 94.22±.12 | 0.9399 | 94.25±.11 | 0.9402 | 98.21±.10 | 0.9814 |
| 0.9 | 96.27±.10 | 0.9612 | 94.40±.19 | 0.9449 | 94.64±.18 | 0.9442 | 98.10±.01 | 0.9805 |

TABLE 5.3: Precision, Recall, and F1 score achieved using SSDAE + SVM model on S-HCR dataset

| Alphabet | Precision | Recall | F1 score | Alphabet | Precision | Recall | F1 score |
|---|---|---|---|---|---|---|---|
| a | 0.99 | 0.99 | 0.99 | n | 0.99 | 0.98 | 0.99 |
| b | 0.97 | 0.99 | 0.98 | o | 0.99 | 0.99 | 0.99 |
| c | 1.00 | 1.00 | 1.00 | p | 1.00 | 1.00 | 1.00 |
| d | 0.99 | 0.99 | 0.99 | q | 0.98 | 0.98 | 0.98 |
| e | 1.00 | 1.00 | 1.00 | r | 0.98 | 0.99 | 0.98 |
| f | 0.99 | 0.98 | 0.99 | s | 0.97 | 0.97 | 0.97 |
| g | 0.98 | 0.98 | 0.98 | t | 0.97 | 0.95 | 0.96 |
| h | 0.99 | 0.99 | 0.99 | u | 0.96 | 0.95 | 0.95 |
| i | 0.97 | 0.98 | 0.98 | v | 1.00 | 1.00 | 1.00 |
| j | 0.98 | 0.98 | 0.98 | w | 0.99 | 0.99 | 0.99 |
| k | 0.99 | 0.99 | 0.99 | x | 1.00 | 1.00 | 1.00 |
| l | 0.99 | 0.97 | 0.98 | y | 1.00 | 1.00 | 1.00 |
| m | 0.98 | 0.97 | 0.98 | z | 1.00 | 0.99 | 0.99 |

#### 5.4.2.1 Analysis of the effect of different corruption rates on SSDAE

Vincent et al. [264,267] reported that using the DAE helps to attain qualitatively better features, resulting in improved classification performance compared to basic auto-encoders for image-based classification tasks. Motivated by this fact, we tried to gain a comprehensive understanding of data corruption on our proposed modified autoencoder (SSDAE).

Experiments were performed by adding varying corruption rates to the SSDAE network and analyzing their impact on the learned features representation. These learned features are given as input to baseline machine learning classifiers, and the

standard performance metrics are evaluated. The varying corruption rates are controlled by the $\eta$ variable.

Figure 5.9 (a) illustrates the mean classification accuracy at different corruption rates using LDA, K-NN (K=3), K-NN (K=5), and SVM algorithms. Corruption rates (ranging from 0.03 to 0.9) were applied while maintaining a constant sparsity parameter of 0.05. In the figure, the horizontal axis represents the corruption rate, whereas the vertical axis indicates the classification accuracy achieved using these baseline algorithms. Additionally, the Matthews Correlation Coefficient (MCC) was calculated to correspond to these corruption rates. Table 5.2 summarizes the mean Matthews correlation coefficient (MCC) and recognition accuracy obtained through ten-fold cross-validation. The MCC score provides better insights when the classifier performs well across all true positive, true negative, false positive, and false negative samples.

Figure 5.9 (a) summarizes that the SSDAE network was able to accommodate the different noise levels without much adverse effects on its performance. Considering the reported performance metric (mean accuracy and MCC) with an increase in the corruption rate, the performance of SSDAE was quite stable. All the conventional machine learning classifiers show a similar stable pattern in terms of reported performance metrics, suggesting the favorable learned feature representation of the HCR task. Moreover, the experimental results depicted in Figure 5.9 (a) and Table 5.2 indicate that the SSDAE + SVM combination outperforms the other baseline algorithms in terms of both accuracy and MCC scores.

During the experiment, corruption operation combined with a sparsity constraint (SSDAE) obtained combined advantages of DAE and SSAE. The goal of this fusion was to achieve a better learned representative feature subset while simultaneously addressing noise issues. The combined sparsity and corruption operation improved the SSDAE model performance.

#### 5.4.2.2 Ablation Study: Justification for added sparsity

In an additional experiment, we replaced the SSDAE with a stacked denoising autoencoder (SDAE) to analyze the effect of removing sparsity constraints on pipeline performance. Figure 5.9(b) and Table 5.2 summarize the mean classification accuracy and MCC score achieved without the sparsity constraints. The network architecture was kept similar to SSDAE in terms of hidden layers and nodes. The neural network was trained using dropout as a regularizer, with a "keep probability" set to 0.9.

Unlike SSDAE, the corruption rate in the SDAE network significantly influences the model's performance. The random noise is induced to corrupt the original data, and eventually, the corrupted data is provided to the neural network to learn to reconstruct the training examples. The varying corruption rate affects the quality of learned feature representations. Figure 5.9(b) shows the effect of corruption rate on the classification accuracy using different baseline classifiers. The figure illustrates that the performance metrics appear more unfavorable with a higher corruption rate. In general, classification accuracy tends to decrease for most algorithms with increasing corruption rates. However, the SVM still attains greater classification accuracy than LDA and K-NN classifiers throughout the variation.

The results from this ablation study further underscore the effectiveness of incorporating data corruption and sparsity constraints in enhancing the performance of the HCR pipeline. Using the noise alone, without sparsity, provided less favorable results. Table 5.4 and table 5.2 highlight the accuracy and the MCC scores achieved while using DSAE and SSDAE networks, respectively. Considering Figure 5.9(a) and 5.9(b), we can compare the classification accuracy obtained using the two optimal models (DSAE +SVM and SSDAE+SVM ) for the same HCR task. The notable difference between the achieved classification accuracy suggests the superiority of combination corruption and data sparsity as an acceptable and efficient

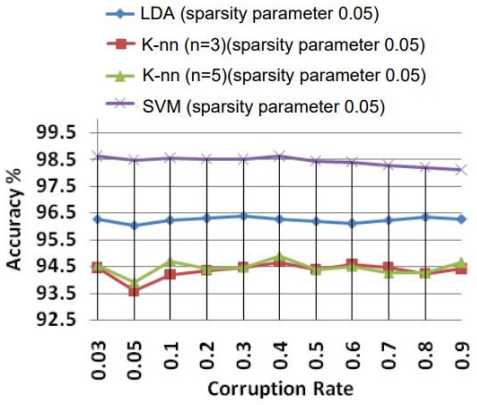
technique for finding representative features for the sEMG-based handwritten character recognition.

#### 5.4.2.3 Feature Visualization

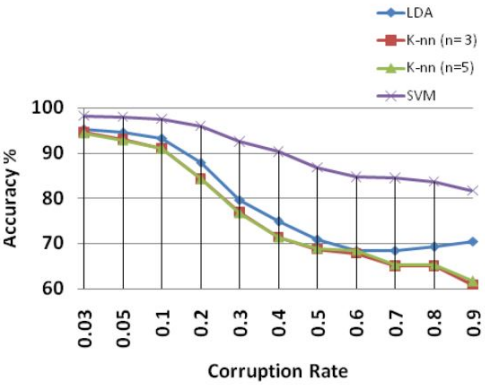
To analyze the effectiveness and the quality of the feature learned, t-distribution Stochastic Neighbor Embedding (t-SNE) [241] was used to visualize the learned features by mapping them into lower two-dimensional space. The t-SNE is a nonlinear dimensionality reduction technique that efficiently captures the local and global structure(pattern) in the original space and can efficiently represent that pattern in the lower dimension [242]as a cluster. It ensures that points close to each other in the higher-dimensional space remain close in the reduced dimensions. Unique colors and symbols are used to represent different class data points, highlighting within-class similarities.

Figure 5.10 illustrates the distribution of the learned features visualized through t-SNE. For plotting the t-SNE, the hyperparameters "max-iteration" and "perplexity" were set to 1000 and 50, respectively. Features corresponding to each of the 26 classes are represented using different colors.

The t-SNE plot in Figure 5.10 shows that feature sets corresponding to different classes form distinct, less overlapping clusters, indicating favorable class separability. Adequate class separability depicts the presence of better class representative features and also facilitates better classification results. So, initial observations from the t-sne plot indicate that the learned feature representation obtained from the SSDAE network contains sufficient discriminatory properties, enabling classifiers to easily differentiate between classes. In a few cases, the t-SNE plot shows some data points clustered with other classes, indicating these points are difficult to identify. A higher three-dimensional projection may be required to accurately predict their exact characteristics.




(a) Graph highlighting the classification accuracy achieved with varying corruption rate with added sparsity constraint using S-HCR dataset




(b) Graph highlighting the classification accuracy achieved with varying corruption rate using S-HCR dataset

FIGURE 5.9: Classification accuracy achieved with varying corruption rates for SSDAE and SDAE on S-HCR dataset

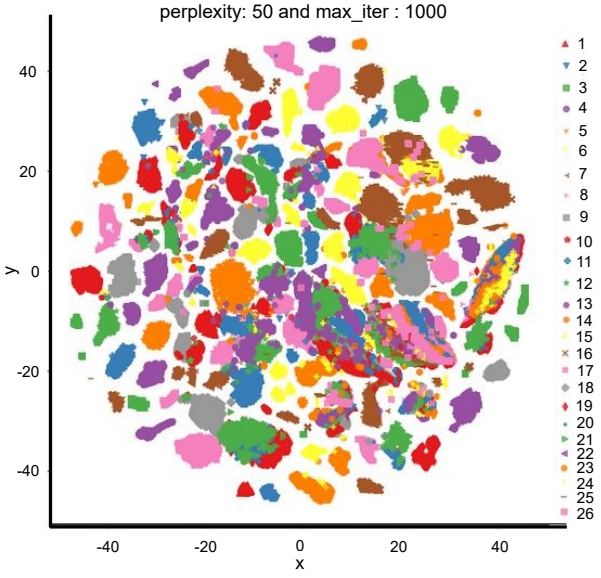



FIGURE 5.10: t-sne plot showing the feature distribution learned using SSDAE network for S-HCR dataset

TABLE 5.4: The classification accuracy and mean MCC score achieved using SDAE network and baseline algorithms

| | LDA | | K-NN(K=3) | | K-NN(K=5) | | SVM | |
|---|---|---|---|---|---|---|---|---|
| $\eta$ | Accuracy | MCC | Accuracy | MCC | Accuracy | MCC | Accuracy | MCC |
| 0.01 | 95.30±.18 | 0.9553 | 94.57±.15 | 0.9429 | 94.46±.11 | 0.9418 | 98.34±.08 | 0.9834 |
| 0.03 | 94.70±.17 | 0.9511 | 93.01±.15 | 0.9437 | 92.84±.11 | 0.9424 | 98.06±.06 | 0.9827 |
| 0.1 | 93.33±.16 | 0.9306 | 91.15±.16 | 0.9079 | 90.98±.20 | 0.9062 | 97.62±.09 | 0.9753 |
| 0.2 | 87.97±.25 | 0.8748 | 84.29±.19 | 0.8366 | 84.24±.21 | 0.8359 | 96.03±.89 | 0.9587 |
| 0.3 | 79.74±.23 | 0.7891 | 76.94±.27 | 0.7610 | 76.81±.27 | 0.7588 | 92.55±.10 | 0.9224 |
| 0.4 | 74.87±.32 | 0.7386 | 71.36±.30 | 0.7049 | 71.41±.32 | 0.7025 | 90.30±.05 | 0.8991 |
| 0.5 | 70.99±.25 | 0.6982 | 68.71±.27 | 0.6745 | 68.84±.27 | 0.6758 | 86.88±.24 | 0.8634 |
| 0.6 | 68.45±.30 | 0.6718 | 67.85±.36 | 0.6656 | 68.45±.26 | 0.6717 | 84.75±.11 | 0.8413 |
| 0.7 | 68.47±.36 | 0.6721 | 65.01±.30 | 0.6360 | 65.38±.32 | 0.6397 | 84.17±.23 | 0.8352 |
| 0.8 | 69.27±.39 | 0.6803 | 64.95±.24 | 0.6354 | 65.35±.25 | 0.6395 | 82.66 ±.11 | 0.8196 |
| 0.9 | 70.44±.44 | 0.6923 | 60.76±.59 | 0.5910 | 61.58±.37 | 0.6002 | 81.61±.20 | 0.8087 |

### 5.4.3 Scalability Analysis

For the S-HCR dataset, we evaluate the scalability and performance of the pipeline by examining the relationships between the learning curve (with training and validation scores), model fit times, and training examples. We plotted different graphs to illustrate these interrelationships.

Model fit time represents the time spent to adjust the model's parameters based on the input data and target values to learn the underlying patterns and relationships, preparing the model for making accurate predictions on new data.

Figure 5.11 depicts the learning curve, with the vertical axis representing the training and validation scores and the horizontal axis representing the number of training examples. With less than 20000 samples, the pipeline achieves a favorable training and validation score above 0.95 and smoothly converges towards the maximum recognition accuracy with increased instances. Such observation suggests that Our pipeline is trained with a reasonable number of samples, and increasing the number of training instances beyond does not adversely impact average recognition accuracy. Figure 5.12 describes the delay incurred (in unit time) to train the model with respect to the number of training instances. If the training time increases linearly with the number of training samples, the model is likely to scale well with larger datasets. However, if the training time increases exponentially, the model

may face challenges in handling larger datasets efficiently. However, for our model, After 60000 samples, the delay incurred (Model fit time) with the number of samples almost has a linear relationship, suggesting its scalability.

Figure 5.13 shows the relationship between the accuracy achieved and the delay incurred. Around 8000 samples, the training and validation scores have the least variation with training samples. Consistent training scores with increasing data size suggest that the model can effectively utilize more data and may generalize well.

Considering the data presented in Figures 5.11, 5.12, and 5.13, we conclude that our pipeline is scalable. The observations indicate that the pipeline can manage additional data without significant performance loss, making it well-suited for real-world applications involving large datasets.

### 5.4.4 Comparison with state of the art methods

In the field of myoelectric-based rehabilitation for upper limb amputees and robotics, it is important to build an efficient control unit for prostheses that can help to replicate human hand movement.

To achieve this functionality, one of the popular options is to deploy pattern recognition algorithms that can easily identify and differentiate between different signal patterns generated corresponding to different hand movements. These pattern recognition algorithms typically learn to model the previously recorded electromyographic activity (EMG) to complex multi-degree of freedom (DOF) movements [268, 269].

This ability of pattern recognition algorithms to model the relationship between signal patterns and hand movements has also been used in other areas, such as online handwriting recognition and reconstruction, the development of neuroprosthetics, and other diagnostic tools.

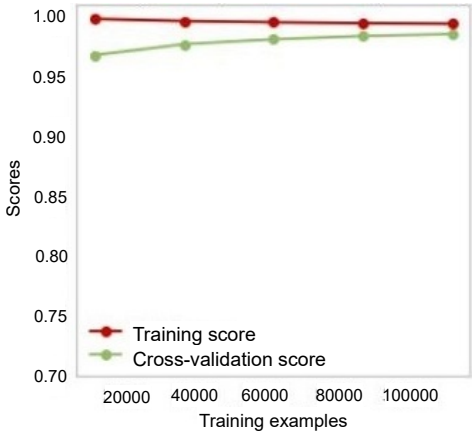



FIGURE 5.11: Learning curve showing training and cross-validation scores for S-HCR dataset using (SSDAE + SVM) model

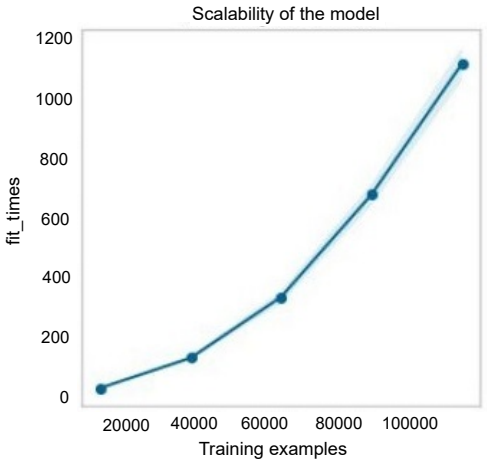



FIGURE 5.12: Plot showing the model training time with respect to the number of training samples for S-HCR dataset using (SSDAE + SVM) model

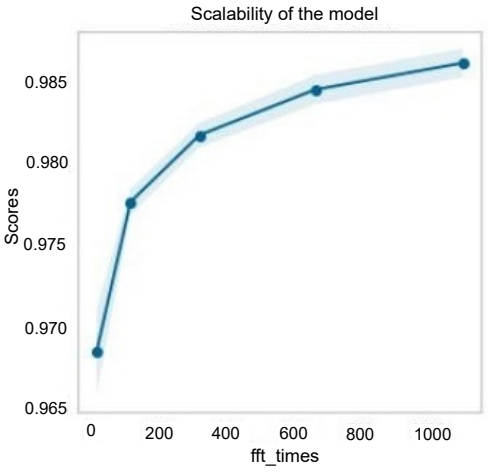



FIGURE 5.13: Graph showing the relationship between the model training time and cross-validation score for S-HCR dataset using (SSDAE + SVM) model

Previous works in this area have established that sEMG and IMU signals can be utilized to classify handwritten characters but still leave room for improvement. Linderman et al. [14] in their work show that the sEMG signals generated during the handwriting activity could be efficiently decoded into font characters and achieved a classification accuracy of 97% for the task. However, their study used a limited character set (0-9 digits) collected from 6 subjects for their evaluation. In another work, Huang et al. [84] applied Dynamic time wrapping(DTW) to classify the ten digits, 10

Chinese characters, and 26 English alphabets in different observations. The dataset collected from three subjects achieved a classification accuracy of 84.29% for the English alphabet. In a similar work based on DWT [85], the authors applied the two-phase template matching approach with a modified Mahalanobis distance, eventually improving the recognition accuracy up to 92.42%. However, their proposed model was less generalized as it was evaluated on a limited number of subjects (four people). Further, Hernandez et al. [86] applied the deep learning architecture for recognizing the English alphabet. Their proposed model achieved the recognition accuracy of 94.85% for the task. The deep learning model used for the task performs automatic feature extraction; hence, these models are less interpretative in terms of features used.

Our proposed model, using sEMG signals from the S-HCR dataset, achieves higher average accuracy compared to the related works mentioned earlier. The DAE characteristics enhance the model's robustness to noise. Table 5.5 provides a comparative analysis of our sEMG-based handwritten character recognition approach with other state-of-the-art methods.

Additionally, we considered IMU-based HCR research for comparison. Wang et al. [270] utilized an accelerometer, achieving a recognition accuracy of 84.8% for numerical digits using a user-independent analysis and a Dynamic Time-Warping algorithm. Similarly, Schrapel et al. [271] used a pen equipped with an IMU unit and a microphone to recognize handwritten digits, achieving a recognition accuracy of 78.4%. Dash et al. [30] employed a similar approach to recognize hand character gestures made in the air using IMU sensors. Using a deep learning architecture consisting of a Convolutional Neural Network and Gated Recurrent Unit, they achieved a classification accuracy of 91.7% for person-independent evaluation.

Most of the IMU-based papers discussed above were limited to numerical digits (0-9), representing only a subset of handwritten characters. Hence, these models do not fully represent generalized handwritten character recognition.

However, some studies have expanded the scope to include a larger set of characters. Wehbi et al. [272] applied a convolutional neural network to recognize both lowercase and uppercase English alphabets, achieving a recognition accuracy of 86.97%. Their work included an additional force sensor for the task. Similarly, Ott et al. [107] used various machine and deep learning models to recognize lowercase and uppercase English alphabet characters. They utilized acceleration, gyroscope, magnetometer, and force signals for the handwriting recognition task, achieving 90% accuracy for writer-dependent evaluations. JJing et al. [273] proposed a model to recognize and reconstruct handwriting movements made in the air. IMU sensors attached to the fingertip were used to recognize ten digits and 26 lowercase English alphabets. Using a support vector machine, their model achieved an 88.0% recognition rate. Zhang et al. [219] utilized smartwatch-based IMU sensors to recognize Chinese characters, achieving a recognition accuracy of 96% with a deep convolutional neural network (DCNN).

For only sEMG-based classification, our recognition accuracy is the highest among the other state-of-the-art approaches. However, it should be noted that Beltran et al. [86] have used more classes (26 alphabets + 10 digits) than our 26 classes. However, our model is more generalized because we have validated it on a more significant number of people (fifteen) than the three people used in their work.Considering the above-discussed research work, our proposed model provides a decent offline recognition accuracy for the Handwritten English alphabet recognition task. The obtained experimental results suggest that the deep feature learning-based classification approach has sufficient potential to be used in online handwriting recognition tasks.However, as shown in Table 5.5, there is room for improvement in the classification accuracy of handwritten character recognition models built solely on sEMG signals.

TABLE 5.5: Comparison of the proposed method for handwritten character recognition with state-of-art approaches

| Paper | Method used | Number of subjects | Number of classes | Sensor used | Recognition Accuracy |
|---|---|---|---|---|---|
| [14] | Template matching | 6 | 10 | sEMG | 97.00% |
| [84] | Dynamic time wrapping | 3 | 26 | sEMG | 84.29 % |
| [85] | Dynamic time wrapping | 4 | 26 | SEMG | 92.42 % |
| [86] | Deep learning (LSTM+RNN) | 3 | 36 | sEMG | 94.85% |
| [271] | Deep neural network | 26 | 10 | Acc and Gyro | 78.40% |
| [270] | Dynamic time warping | 10 | 10 | Acc | 84.80% |
| [272] | Convolutional neural network | 114 | 26 | IMU & Force sensor | 86.97% |
| [107] | CNNs, LSTMs, and BiLSTMs | 119 | 52 | Acc & Gyro | 90.00% |
| [30] | CNNs and GRU | 12 | 10 | Acc & Gyro | 91.70% |
| [273] | Support Vector Machine | – | 36 | IMU sensors | 88.00% |
| **Proposed (S-HCR dataset )** | SSDAE + SVM | 15 | 26 | sEMG | **98.62%** |

#### 5.4.4.1 Run Time Analysis

To approximate the computational complexity, we analyzed the end-to-end processing delay of our proposed pipeline while classifying a single test sample. The pipeline was divided into multiple components, similar to an online classification system. An online classification system consists of modules such as data acquisition (buffer storage) and segmentation (windowing), feature extraction, and a module that uses a trained model for predicting class labels [73].

In our proposed pipeline, the encoding mechanism introduces an additional delay. The input features are encoded using our trained SSDAE model, producing a dense, reduced feature vector representation. This learned feature vector is then classified using baseline classifiers (LDA, K-NN (K=3), K-NN (K=5), and SVM). We defined the time spent by SSDAE to encode a feature vector as the encoding delay ($T_{Enc}$) and the time spent by baseline classifiers to predict the label as the recognition time ($T_{Recog}$).The sum of these time delays is referred to as the Response time. (Explained in detail in Section 2.3.12)

Table 5.7 shows the Response time value for the SSDAE+SVM model on the S-HCR dataset while classifying a single instance of test data, approximated at 0.69178 sec. All experiments were performed on a PC with Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz and 16 GB RAM. Additionally, we calculated the time required to train

the SSDAE network and baseline algorithms on the S-HCR dataset. Table 5.6 highlights the training times for different models, with SSDAE+SVM having the highest training time, followed by SSDAE+LDA. Training times for other models are comparable.

**Low-Cost Device Performance Evaluation** As discussed previously, to evaluate the model's performance on low-cost devices, we deployed our trained SSDAE+SVM model on a Raspberry Pi using TensorFlow Lite. The recognition time on the Raspberry Pi was 0.3473 sec, compared to 0.0156 sec when deployed on a PC with Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz and 16 GB RAM. This demonstrates the feasibility of deploying the model on low-cost hardware, albeit with an expected increase in recognition time due to the lower computational power.

TABLE 5.6: Training time of the different model used for HCR task

| Model | Training time (in seconds) |
|---|---|
| SSDAE+LDA | 15719.671 |
| SSDAE+K-NN (K=3) | 15718.318 |
| SSDAE+K-NN (K=5) | 15718.240 |
| SSDAE + SVM | 16441.725 |

TABLE 5.7: Response time of the proposed pipeline to identify a single handwritten lowercase English alphabet using SSDAE+SVM model on S-HCR datset

| Window segmentation ($T_{w_s}$) | Feature Extraction ($T_{fe}$) | Encoding Delay ($T_{Enc}$) | Recognition Time ($T_{Recog}$) | Response Time ($T_{w_s}$+$T_{fe}$+$T_{Enc}$+$T_{Recog}$) |
|---|---|---|---|---|
| 0.6 sec | 0.02928 sec | 0.046875 sec | 0.015625 sec | 0.69178 sec |

#### 5.4.4.2 Application: Exploring Potential Use-Case Scenarios

The proposed classification model has the potential to be used in the digital storage of handwritten text without traditional input devices(such as a keyboard). In this scenario, an individual writes text while equipped with sEMG/IMU sensors on their forearm. These sensors transmit signals wirelessly to a computer-based recognition tool, which processes the signals, identifies the characters, and inputs the information into word-processing software. This system is particularly advantageous

in smart classroom environments, where handwritten notes are often preferred over typing [274]. Additionally, a teacher's handwritten notes on a whiteboard can be efficiently digitized using this classifier.

The efficiency and scalability of this system can be significantly enhanced through integration with IoT infrastructure. The cloud maintains a synchronized replica of the trained machine-learning model. During the writing process, signals are transmitted simultaneously to both the cloud and the local digitization tool. The local tool processes the real-time handwritten characters, while the cloud stores the signals for continuous training and updating of the machine-learning model. The updated model can be downloaded as needed, ensuring ongoing improvement and adaptation. The system's architecture is illustrated in Figure 5.14.

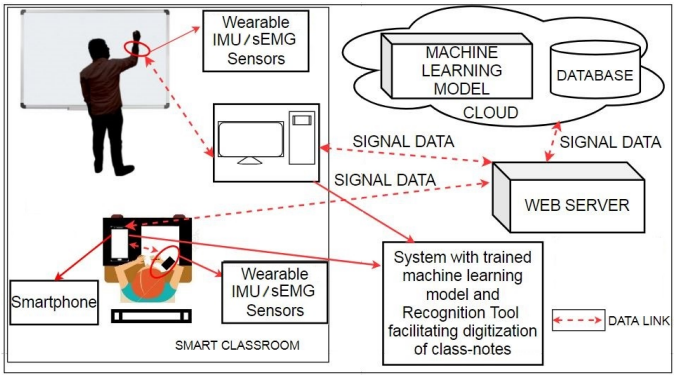



FIGURE 5.14: A diagram illustrating a real-world smart classroom scenario where the proposed handwriting recognition model can be deployed.

### 5.4.5 Threats to Validity

The performance of the proposed model may be affected by several types of validity concerns, outlined below.

#### Internal Validity

To keep data collection consistent, sEMG sensors were placed in fixed positions on the forearm, and the gesture-writing task used a standardized font size with grid

lines. While these controls reduced variation during the experiment, the model's performance may drop in real-world settings where such controls are not possible.

#### External Validity

The study only included right-handed participants, which limits how well the results apply to left-handed users or tasks involving both hands. Future work should include a more diverse group of users and test the system in less controlled conditions to improve generalizability.

#### Construct Validity

The model was trained on dynamic gesture-writing tasks using a pen and paper, which involves natural hand movement. However, it does not capture variations in object type, grip force, or task intent. This may limit how well the model reflects real-world hand grasp behavior.

#### Conclusion Validity

The effects of data collected at different times of day or across days were not separately analyzed. These factors could affect how stable the model is over time. Future studies should explore this to better understand the system's reliability.

## 5.5 Summary

In this chapter, we explored the potential of leveraging deep feature learning for sEMG-based dynamic hand gesture recognition, specifically focusing on handwritten character gestures. The research was driven by the need to develop a robust and efficient Handwritten Character Recognition (HCR) system that can effectively harness muscle activities for enhanced accuracy.

The proposed solution involved using a stacked sparse denoising autoencoder (SSDAE) network for deep feature extraction. This approach capitalized on the

strengths of both sparse and denoising autoencoders, enabling the learning of compact and robust feature representations. Through rigorous experimentation, we demonstrated that the SSDAE network could efficiently extract meaningful features from sEMG signals, leading to significant improvements in classification performance.

The results obtained from extensive evaluations validated the effectiveness of the proposed pipeline, achieving a remarkable recognition accuracy of 98.72% even when using traditional machine learning classifiers. This performance underscores the potential of deep feature learning in transforming sEMG signals into a viable medium for online handwriting recognition and real-time Human-Computer Interaction (HCI) applications.

Additionally, the chapter addressed various challenges associated with sEMG-based recognition, such as noise susceptibility and feature extraction complexity. The integration of denoising autoencoders proved instrumental in enhancing noise robustness, while the introduction of sparsity constraints facilitated the learning of more representative features.

Our proposed method was further compared with state-of-the-art approaches in sEMG-based HCR research, demonstrating superior recognition accuracy and generalizability across multiple subjects. The scalability and efficiency of the pipeline were confirmed through runtime analysis and performance evaluation on low-cost devices, highlighting its applicability in real-world scenarios.

In conclusion, this chapter presents a comprehensive framework for sEMG-based dynamic hand gesture recognition using deep feature learning. The findings not only contribute to the field of HCR but also pave the way for future research and development of innovative solutions in handwriting recognition and beyond.