SN Computer Science (2024) 5:273
https://doi.org/10.1007/s42979-023-02482-w

ORIGINAL RESEARCH

Developing an Offline and Real-Time Indian Sign Language Recognition System with Machine Learning and Deep Learning

K. Priya[1] · **B. J. Sandesh**[1]

Received: 25 October 2023 / Accepted: 7 November 2023

Abstract
Sign language is a powerful form of communication for humans, and advancements in computer vision systems are driving significant progress in sign language recognition. In the context of Indian sign language (ISL), early research focused on differentiating a limited set of distinct hand signs, often relying on specialized hardware such as sensors and gloves, also most of the works were experimented on the dataset captured under controlled environments. This research aims to enhance communication for the speech and hearing impaired community by recognizing static images of ISL digits and alphabets in both offline and real-time scenarios. To achieve this, two publicly available datasets were used, containing a total of 42,000 sign images and 36,000 static signs, respectively. The dataset1 consists of sign images that were taken under controlled environments, whereas the dataset2 consists of sign images that were taken in different environments with varying backgrounds and lighting conditions. Dataset1 was experimented with and without using preprocessing techniques, while dataset2 underwent similar testing. We employed both machine learning and deep learning with CNN to categorize the ISL alphabets and numbers. In the machine learning approach, image preprocessing techniques such as HSV conversion, skin mask generation, and skin portion extraction and Gabor filtering were used to segment the region of interest, which was then fed to five ML models for sign prediction. In contrast, the DL approach used CNN model. In addition, probability ensemble testing was performed on both datasets to compare the accuracies. Real-time recognition was also conducted using a custom dataset, employing the YOLO-NAS-S model. This study contributes to the advancement of ISL recognition by conducting a comparative analysis of ML algorithms and CNNs, examining their performance with and without preprocessing techniques.

Keywords Indian sign language · Machine learning · Deep learning · CNN · HSV · Skin mask · Gabor filters · YOLO-NAS-S

Introduction

Sign language is a visual language that involves the use of hand movements and facial expressions to communicate ideas. Individuals who have hearing loss use this natural language to express their thoughts and emotions, using hand movements to convey letters, words, and even entire phrases. This method of communication is vital in enabling those with hearing impairments to express themselves and facilitating effective communication between them and others. However, it is important to note that there are many different sign languages used around the world, each with its unique characteristics that are influenced by the country's culture and spoken language. While there are various native languages spoken in different regions and communities, translation is often necessary to enable effective communication between them. The speech and hearing impaired community, in particular, use gestures, mimics, signs, and expressions to communicate with each other and with others. However, these gestures may not always correspond directly to the intended term or alphabet, leading to a significant communication barrier between those who are hearing and those who are not. The symbols utilized in sign language exhibit a wide range of complexity and comprise

This article is part of the topical collection "Advances in Computational Approaches for Image Processing, Wireless Networks, Cloud Applications and Network Security" guest edited by P. Raviraj, Maode Ma and Roopashree H R.

✉ K. Priya
priyak@pesu.pes.edu

B. J. Sandesh
sandesh_bj@pes.edu

[1] Computer Science and Engineering Department, PES University, Bangalore, India

Published online: 20 February 2024

various hand movements, including representations that use both the hands. Static symbols are commonly used to represent letters. A real-time system can be created to eradicate communication barriers for individuals who are deaf or hard of hearing using computer vision technology. Such a system can be adapted to any language. However, developing an efficient and accurate system has been a challenge. Previous works have used hand-crafted features but have limitations and require specific conditions. The majority of these studies rely on techniques for pattern recognition and feature extraction, such as HOG, SIFT, and LBP. However, relying solely on a single feature is often insufficient, and hybrid approaches have been introduced to address this issue. For real-time applications, faster methods are needed. With the advancement of computing technology, computers now have faster processing speeds through parallel implementation. While traditional problem-solving techniques may use a single core, GPU systems offer a greater number of cores for parallel computing. Deep learning technique such as convolutional neural networks (CNNs) can be employed to develop a self-learning system tailored to our requirements. CNNs are a popular form of deep learning that can address a wide range of computer vision problems. ISL uses two hands for communication which can lead to obscuring of features. There is also a lack of datasets and variance in sign language across different regions in India, which has limited efforts in ISL gesture detection. According to the 2011 census, there are nearly 2.68 Cr (around 2.21% of the total population) deaf and dumb individuals in India out of which deaf people constitute 18.9% and mute people constitute 7.5% of the disables population. It is extremely disheartening to see the gap that this has created in the Indian society. This inspired us to focus our research on ISL Alphabets and numbers as shown in Fig. 1.

In recent years, there have been efforts to promote the use of Indian sign language (ISL) and improve access to education and other services for the deaf community in India. As per reports, under the National Education Policy 2020, there is a plan to standardize ISL across the country to ensure that students with hearing disabilities have access to quality education. Prime Minister Narendra Modi has also highlighted the importance of promoting ISL and providing equal opportunities for people with disabilities in various speeches and events. There are also initiatives to develop technology and tools to support ISL, including apps and software for translation and interpretation.

Related Work

Computer vision, a technology with versatile applications, is being increasingly used in AI-based systems such as robotics, self-driving vehicles, and marketing. Its significance lies in its ability to effectively address problems related to object identification and image classification.

Fig. 1 ISL—Indian sign language (alphabets and numbers) source: Google

Divya Deora et al. [1] have proposed a framework for a sign language recognition system in their paper, which utilizes PCA (Principal Component Analysis) for recognition and glove-based color segmentation. However, the proposed method faced difficulties with overlapping and motion. Recent research in this field has focused on static ISL signs [2] captured under controlled conditions using hardware devices. A variety of methods have been used to prepare the data for analysis, such as Otsu's thresholding [3, 4], motion-based and skin color segmentation, as well as background elimination [5]. Wavelet decomposition, Fourier descriptors, and scale-invariant feature transform have all been used to extract features. A variety of classifiers have also been implemented, such as Fuzzy systems, KNN (K-Nearest Neighbor), HMM (Hidden Markov Models), SVM (Multiclass Support Vector Machines) and ANN (Artificial Neural Networks) [6].

Another study by Purva A et al. [7] suggests an edge detection method for recognizing hand gestures, in which features are collected and matched using nearest neighbor classifiers and template matching. A gesture recognition method employing the Microsoft Kinect sensor was introduced by Reheja [8], which increases accuracy by leveraging RGB-D images. In order to recognize Indian sign language, PranaliLoke [9] created an Android-based system. A neural network is used to train the system, and the Sobel operator is used to extract features.

Another recognition system was developed by Beena M.V. et al. using depth images obtained from the Kinect sensor in their research [10]. The ANN was used to extract features after the system had been trained with 1000 images for each numerical sign. A GPU was used to speed up the training process, which produced results with an accuracy

of 99.46%. A CNN with softmax classification was used to categorize 33 static signs from Kinect depth images as a subsequent extension to their work. The study demonstrated that hand-crafted features became insufficient for classification as the number of classes increased. However, the CNN structure was able to learn from the training set and achieved higher accuracy than other traditional methods.

Otiniano et al. [11] employed scale-invariant feature transform descriptors for the RGB images and Gradient Kernel Descriptors for the depth images to separate the hand regions from the background. The Support Vector Machine was then given the combined data, and it was able to categorize signs with 90.2% accuracy. Recurrent 3D convolutional neural network (R3DCNN) for dynamic gestures was employed by Molchanov et al. [12] along with temporal classification to recognize the dynamic gestures. Data were collected using stereo-IR, RGB, and depth sensors. Since the R3DCNN model had already been pre-trained using the Sport-1M [13] dataset, they used this transfer learning approach.

In order to recognize hand gestures, Okan et al. [14] proposed a hierarchical CNN-based architecture that made use of two convolutional neural networks: a large 3DCNN and a simple CNN. On the EgoGesture benchmark, this model's classification accuracy was 94.04%, while on the NVIDIA test, it was 83.82%. Similar to this, Xiaokai et al. [15] created a method for reducing dynamic gesture-containing videos into a compact framework that captures its spatio-temporal information for dynamic gesture recognition. They used a Dense Image Network (DIN). A CNN is then used to increase feature extraction from the DNN's output. With the use of a customized ISL dataset and a CNN, Mukesh [16] was able to obtain a classification accuracy of 85.51%. Real-time testing on this model, however, was not done.

The three key stages in SLR are feature extraction, temporal modeling, and prediction. In earlier research, spatial representation was produced using manually created features such HOG-based features [17, 18], SIFT-based features [19, 20], and frequency domain features [21, 22]. Hidden Markov Models (HMM) [23–25] and hidden conditional random fields [26] were used to represent temporal data. Different frame rates were handled in various studies using dynamic time wrapping [27, 28]. Models like Support Vector Machine (SVM) [29], which were used in the prediction phase as a classification problem, were used to predict the signs. On small datasets of less than 100 words, the majority of traditional models for sign language recognition are tested [30–32]. With the addition of deep neural networks, the performance of several video-based tasks, including action recognition [33, 34] and gesture recognition [35], has greatly increased and both have a similar problem structure.

Several methods for automatic identification of sign language gestures have been proposed. Rokade [36] proposed a technique for identifying finger spelling in Indian sign language using skin color segmentation to determine the shape of the sign. Feature extraction was performed using Hu's moment, and SVM and ANN were used for classification. The method achieved high accuracy, with SVM and ANN achieving accuracies of 94.37% and 92.12%, respectively. Katoch et al. [37] proposed a video-based technique for recognizing Indian sign language letters and numerals, using background removal and skin tone segmentation. Classification was performed using CNN and SVM, and the authors utilized the "Bag of Visual Words" (BOVW) model with SURF feature extraction. The method achieved high classification accuracies of 99.17% and 99.64% using SVM and CNN, respectively. Finally, Shenoy et al. [38] proposed a method for recognizing static hand poses in ISL, using skin color segmentation to detect and track hands. Feature extraction was performed using a grid-based approach, and KNN and HMM were used for classification, achieving high accuracies of 99.7% and 97.23%, respectively.

We have used two methods in our study to categorize the signs: the first method used machine learning, with and without image preprocessing, and the second method used deep learning using a CNN model, again with and without preprocessing methods.

System Design and Rationale

This section includes two parts: machine learning and deep learning approach. The machine learning approach involves following stages: data acquisition, data preprocessing, five machine learning algorithms, training, and testing phase. On the other hand, the deep learning approach includes data acquisition, CNN architecture component, model training, and testing as shown in Fig. 2.

Machine Learning Approaches and CNN with Preprocessing for Dataset1

A ML approach refers to the use of ML algorithms and models to analyze large amounts of data and uncover patterns, relationships, and insights. The system uses this knowledge to make predictions or decisions about new data it encounters in the future.

Data Acquisition

Dataset 1—The dataset1 used in this study consisted of 42,000 images that were distributed across 35 different classes and the samples are shown in Fig. 3. The original size of each image was $128 \times 128 \times 3$, but they were later resized to $96 \times 96 \times 3$ to facilitate processing. These images were captured under controlled environments, ensuring

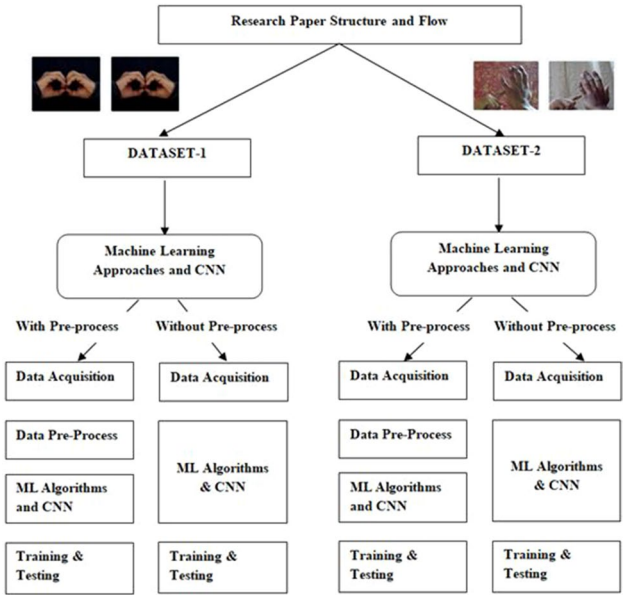

Fig. 2 Structure and flow of the paper

consistent background and lighting conditions throughout as shown in Fig. 3.

Data Preprocess

The purpose of image preprocessing was to improve the quality and suitability of images for further analysis and processing, so that our algorithms can better detect patterns and make accurate predictions. This includes operations such as resizing, HSV Conversion, Skin Mask Generation, Skin Portion Extraction and Normalization.

HSV Conversion HSV is an acronym that stands for hue, saturation, and value, and it refers to the three color channels used in digital image processing. In the HSV color model, hue is used to describe the color shade or tint, saturation represents the intensity of the color, and value indicates the brightness. We used Python implementation of OpenCV library to perform a color space conversion on an input image from blue–green–red (BGR) to hue-saturation-value (HSV) format.

Skin Mask Generation Skin mask generation is a method used to segment skin pixels in an image. Here, we performed skin color segmentation which creates a binary mask containing only the pixels in the image that fall within a specified range of skin color. The range of skin color is defined by lower and upper thresholds in the HSV color space, which are specified using numpy arrays. The binary mask is then processed using morphological erosion and dilation operations with a structuring element to remove small noise and smooth the edges of the mask. This resulting mask was used to detect regions of skin color in the image and also to remove skin color from the image.

Fig. 3 Sample images of dataset1—class 'P'

Skin Portion Extraction Skin portion extraction has a pipeline designed to detect and highlight skin color regions in an image. The pipeline consists of several steps, including converting the input image from the BGR color space to the HSV color space, creating a binary mask to isolate the skin color regions, applying a median blur filter to the binary mask to remove noise. We used binary mask to separate the skin color regions from the input image, the skin color regions are then merged back into the input image to create a final output image that emphasizes the skin color regions. The binary mask is then used again to remove any non-skin pixels that may have been added during the merging operation. Our objective was to convert the image data into tabular numerical data. So for each sample, any array value less than 1 was designated as 255, while any value greater than 1 was assigned a value of 0.

All these preprocessed images can be found in Fig. 4.

Machine Learning Algorithms

The approach used in this research involves training 5 machine learning models with the ISL dataset, which contains 35 classes, with each class containing 1200 images. The performance of each model is then evaluated and compared in the results analysis section.

1. KNN—K-nearest neighbor classifier
2. DT—Decision tree classifier
3. RF—Random forest classifier
4. ET—Extra tree classifier
5. GNB—Gaussian Naïve Bayes classifier

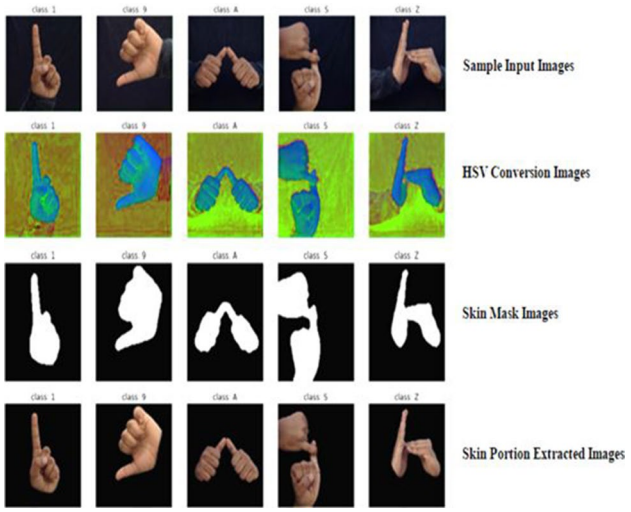

Fig. 4 Preprocessed images

In order to test different classifiers on our two datasets and to examine their performance, we have combined ML techniques. As a probabilistic classifier, we first chose Gaussian Naïve Bayes (GNB), which is known for its efficient computation of high-dimensional data. Next, we chose the non-parametric K-Nearest Neighbors (KNN) technique, which can handle local structure within the data. Random Forest (RF), an ensemble learning technique that incorporates several Decision Trees, was also included. We added Decision Tree (DT) to handle both categorical and continuous features. Finally, as it provides more feature randomization by default, we selected ExtraTrees (ET), another ensemble model.

K-Neighbors Classifier KNN works by identifying k closest neighbors of a given data point and categorizing the point according to the majority class among its neighbors. The parameter n_neighbors is a key setting algorithm, indicating the number of neighbors to consider for classifying a data point. For this particular implementation, n_neighbors has been set to 1500, indicating that the algorithm will consider the 1500 Nearest Neighbors when making classification decisions. The accuracy achieved by the K-Neighbors Classifier model is 93.24%. KNN Classifier equation is shown in (1) as follows:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{1}$$

Decision Tree Classifier The Decision Tree Classifier algorithm is employed for classification tasks and determines the best split for internal nodes based on a set of input features. The "min_samples_split" option outlines the bare minimum of samples necessary to divide an internal node in a Decision Tree. If the total number of samples at a node is less than the specified value of min_samples_split, then the node will not be split. This parameter helps prevent overfitting. In this instance, min_samples_split is configured to 1000, indicating that a node must contain a minimum of 1000 samples to be eligible for splitting. In this scenario, min_samples_leaf is set to 700, implying that each leaf node must have at least 700 samples. The accuracy attained by the Decision Tree Classifier model is 99.42%. Entropy using the frequency table of one attribute is shown in Eq. (2):

$$E(S) = \sum_{i=1}^{c} -p_i log_2 p_i \tag{2}$$

Random Forest Classifier The Random Forest algorithm is a powerful ensemble learning method that combines multiple Decision Trees to make predictions. Here, we have chosen this classifier because, RF works on bagging principle also known as bootstrap aggregation and the final output is based on majority voting. Upon initialization, RF is configured with the following hyperparameters: n_estimators is a parameter determines the number of Decision Trees to be generated by the Random Forest algorithm. In this scenario, it has been set to 1, implying that only one Decision Tree will be employed. When building a Decision Tree, the min_samples_split parameter defines the bare minimum of samples needed to split a node. In this case, it has been set to 100, meaning that a node must contain a minimum of 100 samples to be eligible for splitting. The Decision Tree's min_samples_leaf parameter is set to 100, implying that each leaf node in the Decision Tree will have at least 100 samples. The accuracy achieved by the RandomForestClassifier model is 97.44%. RF Classifier is shown in Eq. (3):

$$\text{MSE} = \frac{1}{N}\sqrt{\sum_{i=1}^{N} (f_i - y_i)^2} \tag{3}$$

where N is the total number of data points, f_i is the model's output value, and y_i is the data point's actual value.

ExtraTree Classifier It is a Decision Tree-based model that combines several trees, and it is designed to prevent overfitting and improve the model's accuracy. The parameter n_estimators specifies the number of trees employed in the ensemble. In this situation, only one tree will be utilized. In this scenario, a node must have a minimum of 100 samples to be eligible for splitting. The accuracy achieved by the ExtraTree Classifier model is 96.09%. Equation (4) is for ET Classifier:

$$\text{Entropy}(S) = \sum_{i=1}^{c} -p_i \log_2(p_i) \tag{4}$$

The equation represents the calculation of entropy in a Decision Tree, where pi represents the proportion of rows

that belong to class label i, and c represents the total number of distinct class labels.

Gaussian NB Classifier The Gaussian Naïve Bayes (GNB) method is a probabilistic machine learning technique that makes use of the Bayes theorem to calculate the probability that a sample will fall into a certain class based on the values of its features. In this particular case, the "var_smoothing" hyperparameter is set to 10,000, indicating a significant amount of smoothing is applied to the variances. This level of smoothing can heavily smooth the variances of the features, making the model more data-sensitive and potentially improving its accuracy. Equation (5) shows Gaussian NB Classifier:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y{}^2}}\exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right) \quad (5)$$

The Gaussian NB Classifier model achieved an accuracy of 95.93%.

CNN Architecture with Preprocessing

In this system, we designed CNN Components as shown in Table 1 to classify Indian sign language.

Table 1 The proposed CNN system architecture with preprocessing

Layer (type)	Output size	Param#
convId_1 (ConvID)	(None, 9214, 64)	256
convId_2 (ConvID)	(None, 9214, 64)	12,352
max_poolingId_1 (MaxPoolingID)	(None, 4606, 64)	0
convId_3 (ConvID)	(None, 4606, 128)	24,704
convId_4 (ConvID)	(None, 4606, 128)	49,280
max_poolingId_2 (MaxPoolingID)	(None, 2301, 128)	0
convId_5 (ConvID)	(None, 2299, 256)	24,704
convId_6 (ConvID)	(None, 2297, 256)	49,280
max_poolingId_3 (MaxPoolingID)	(None, 1148, 256)	0
convId_7 (ConvID)	(None, 1146, 512)	24,704
convId_8 (ConvID)	(None, 1144, 512)	49,280
max_poolingId_4 (MaxPoolingID)	(None, 572, 512)	0
batch_normalization_1 (BatchNormalization)	(None, 572, 512)	2048
flatten_1 (Flatten)	(None, 292,864)	0
dropout_1 (Dropout)	(None, 292,864)	0
dense_1 (Dense)	(None, 512)	149,946,880
dense_2 (Dense)	(None, 35)	17,955
Total Params	151,529,571	
Trainable Params	151,528,547	
Non-trainable Params	1024	

We have created convolutional layers with 64 filters in the neural network architecture to analyze the input data and produce output shapes of (None, 9214, 64) and (None, 9212, 64). These layers detected spatial patterns in the input and extracted them. In order to facilitate spatial dimension reduction to half, a max pooling layer was added. This produced an output shape of (None, 4606, 64). On input data of forms (None, 4604, 128) and (None, 4602, 128), similar convolutional layers with 128 filters were used. The design had layers with 256 and 512 filters, among other filter counts. To improve training stability, a batch normalization layer normalized activations from the preceding layer. The incoming data was then transformed into a 1D array using a flatten layer. The final fully connected layer with 35 units represents the output classes of the classification task. The total number of trainable parameters in the network is 151,528,547, which are optimized during training, while the non-trainable parameters (1,024) are typically associated with certain layers, such as batch normalization, which have fixed parameters.

Training and Testing

The images in the dataset1 underwent various preprocessing techniques, including HSV, skin mask generation, and skin portion extraction. After processing, the images were divided into two sets: 70% of the images, which equates to 29,400, were used for training, while the remaining 30% (12,600 images) were set aside for testing purposes. To

construct the machine learning models, SKlearn library models were imported, and the training dataset was used. Once the models were created, the remaining 30% test dataset was used to evaluate their accuracy, precision, F1-score, and recall value.

The Decision Tree Classifier model has demonstrated an impressive level of accuracy, achieving a score of 99.42%. In fact, when compared to other machine learning algorithms, it has been shown to outperform them significantly. ML models result comparison will be discussed in "Experimentation and Results".

A 70:30 train-to-test split ratio was used to train and assess the CNN model. An accuracy of 99.99% was attained during the testing phase after the data had been preprocessed.

Machine Learning Approaches and CNN Without Preprocessing for Dataset1

The original dataset was loaded, and the images were transformed into Numpy arrays to get them ready for classification models. The scikit-learn train_test_split function split the data and labels into training and testing sets using an 80:20 ratio while reshaping the data array into a flattened form.

Each image was reshaped to become a flattened vector. The data had the following shape: (42,040, 112, 112, 3), which represented 42,040 samples with 112×112 pixel size and 3 RGB channels. A 1D array of form (42,040) was used to represent the labels, which corresponded to the labels for each sample in the data array. The dataset was then tested using 5 ML classifiers described in previous section without any preprocessing techniques, and the results were outlined in the results and analysis section. The Gaussian Naïve Bayes Classifier model has demonstrated an impressive level of accuracy, achieving a score of 95.28% compared to others.

For CNN, 80:20 train-to-tests split ratios was used to train and assess the CNN model. An accuracy of 94.54% was attained during the testing phase without preprocessing the data. Table 2 shows CNN architecture without preprocessing.

Machine Learning Approaches and CNN with Preprocessing for Dataset2

Dataset2 has a collection of 36,000 RGB images representing various static signs. Each sign class in the dataset is represented by 1000 images. These images were taken in uncontrolled environments with different backgrounds and varying illumination conditions. This variability in the

Table 2 The proposed CNN system architecture without preprocessing

Layer (type)	Output size	Param#
Conv2d_1 (Conv2D)	(None, 112, 112, 64)	1792
batch_normalization1	(None, 112, 112, 64)	256
max_pooling2d_1	(None, 56, 56, 64)	0
dropout_1	(None, 56, 56, 64)	0
Conv2d_2 (Conv2D)	(None, 56, 56, 128)	73,856
batch_normalization2	(None, 56, 56, 128)	512
max_pooling2d_2	(None, 28, 28, 128)	0
dropout_1	(None, 28, 28, 128)	0
Conv2d_3 (Conv2D)	(None, 28, 28, 256)	295,168
batch_normalization3	(None, 28, 28, 256)	1024
max_pooling2d_3	(None, 14, 14, 256)	0
dropout_2	(None, 14, 14, 256)	0
Conv2d_4 (Conv2D)	(None, 14, 14, 512)	1,180,160
batch_normalization4	(None, 14, 14, 512)	2048
max_pooling2d_4	(None, 7, 7, 512)	0
dropout_3	(None, 7, 7, 512)	0
flatten_3 (Flatten)	(None, 25,088)	0
dropout_19	(None, 25,088)	0
dense_1 (Dense)	(None, 512)	12,845,568
dense_2 (Dense)	(None, 35)	17,955
Total Params	14,418,339	
Trainable Params	14,416,419	
Non-trainable Params	1920	

Fig. 5 Dataset sample images

environmental conditions adds complexity to the dataset as shown in Fig. 5.

For dataset2, the machine learning techniques that had previously been employed for dataset1 were used again with preprocessing using Gabor filtering method. The Random Forest (RF) model identified 91% of the samples in dataset2 correctly, according to its accuracy score of 91%. Similar to the RF model, the K-Nearest Neighbors (KNN) method performed somewhat better, with an accuracy of 92%. While the Gaussian Naïve Bayes (GNB) method and the Decision Tree (DT) algorithm both had 88% accuracy rates, respectively. On dataset2, however, the ExtraTrees (ET) method stood out as it attained a remarkable accuracy of 99%. This shows that the ET algorithm handles the complicated nature of dataset2 effectively. 80:20 train-test split was maintained throughout the implementation.

The CNN architecture used in this study is same as the one discussed previously for dataset1. In all, there are 14,606,756 parameters in the model, 14,604,836 of which are trainable, and 1920 of which cannot. The dataset was divided into training and testing splits in an 80:20 ratio. Preprocessing methods were used to give the CNN a test set accuracy of 98.60%. 5 ML models performance can be found in Fig. 6.

Machine Learning Approaches and CNN Without Preprocessing for Dataset2

Without preprocessing the data, we obtained 94% for the RF Classifier, 92% for the KNN, 72% for the DT, 96% for the ET, and 19.79% for the GNB.

For CNN model, we used an 80:20 train-test split ratio, and the CNN was trained without preprocessing obtained 97% accuracy on the test set. A total of 14,569,252 parameters make up the model architecture, of which 14,567,332 are trainable and 1920 are not.

The architecture and data processing methods used for the CNN and ML are identical to those employed for Dataset1. ML Models Performance comparison for dataset2 without preprocessing is in Fig. 7.

Proposed System Flowchart

The overall proposed system architecture for both the datasets can be found in Fig. 8. Using our convolutional neural network architecture, the proposed model is trained on a Tesla T4 graphics processing unit (GPU) with 12 GB of memory, 25.45 GB of random access memory (RAM), and a 25.38 GB SSD (solid-state drive).

Experimentation and Results

In accordance with the details provided in the previous section, we conducted our experiments and have summarized the results in Table 3. Our evaluation involved comparing the performance of 5 different algorithms, namely KNN, DT, RF, ExtraTree, and Gaussian NB, for the classification of Indian sign language. We assessed the algorithms' results using precision, recall, F1-score, and accuracy. In addition, we employed both a machine learning (ML) approach and a deep learning (DL) approach, with the ML approach consisting of the aforementioned algorithms and the DL approach employing convolutional neural networks (CNN).

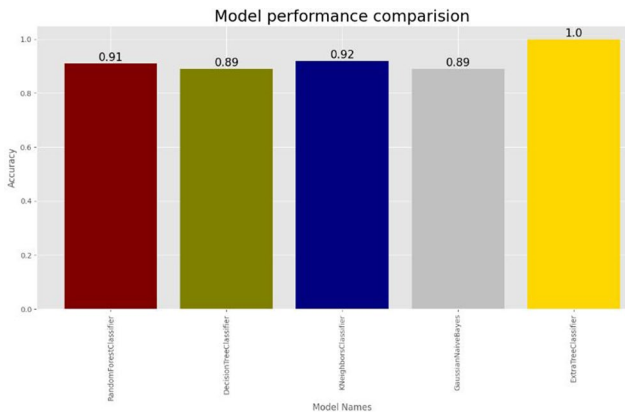

Fig. 6 ML models performance comparison for dataset2 with preprocessing

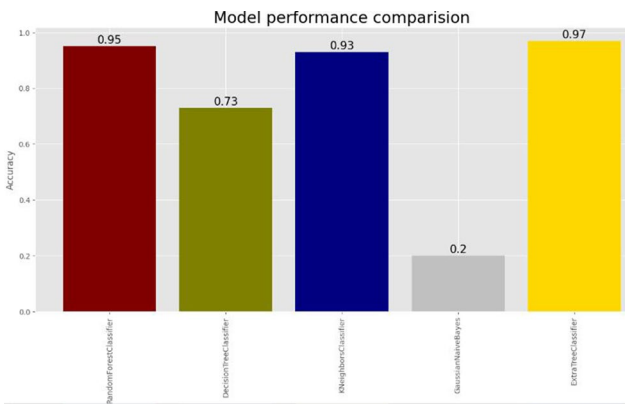

Fig. 7 ML models performance comparison for dataset2 without preprocessing

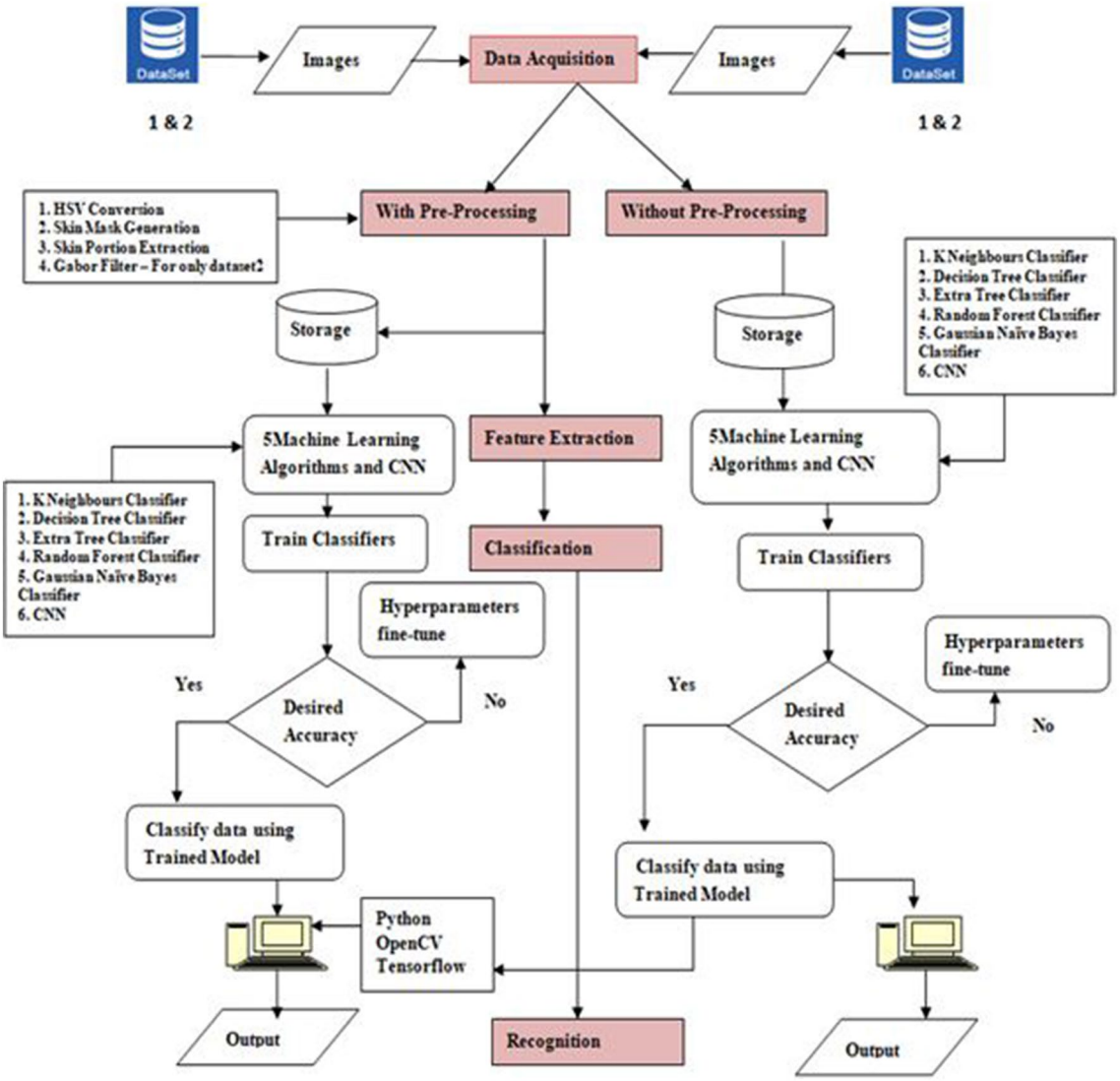

Fig. 8 Proposed system flowchart

Table 3 Accuracy comparison for dataset1 with and without preprocessing

Dataset—1		
Classifiers	With preprocessing accuracy (%)	Without preprocessing accuracy (%)
Random Forest (RF)	97.44	91.60
K-Nearest Neighbor (KNN)	93.24	91.06
Decision Tree (DT)	99.42	91.84
ExtraTree (ET)	96.09	94.86
Gaussian Naïve Bayes (GNB)	95.93	95.28
CNN	99.99	94.54

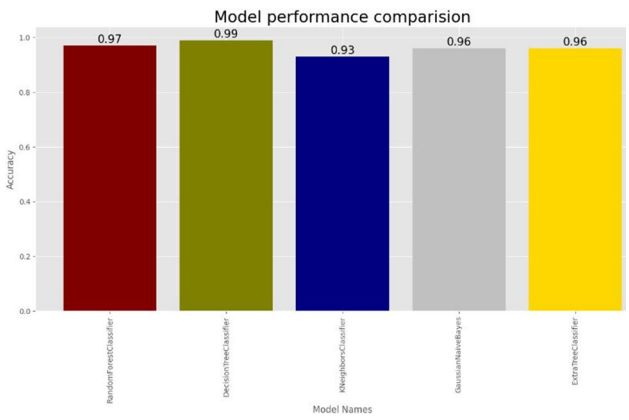

Fig. 9 ML models performance comparison for dataset1 with preprocessing

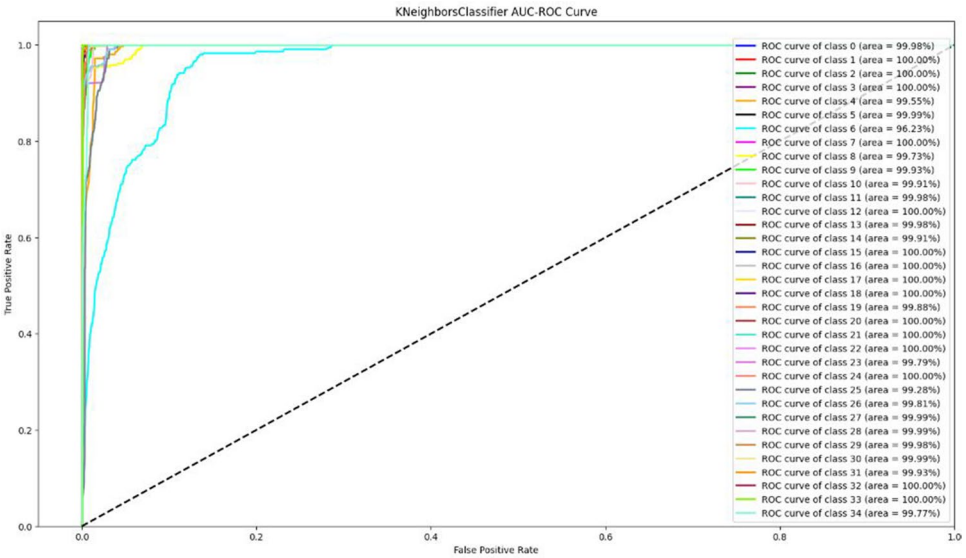

Fig. 10 ROC curve for CNN with data preprocessing

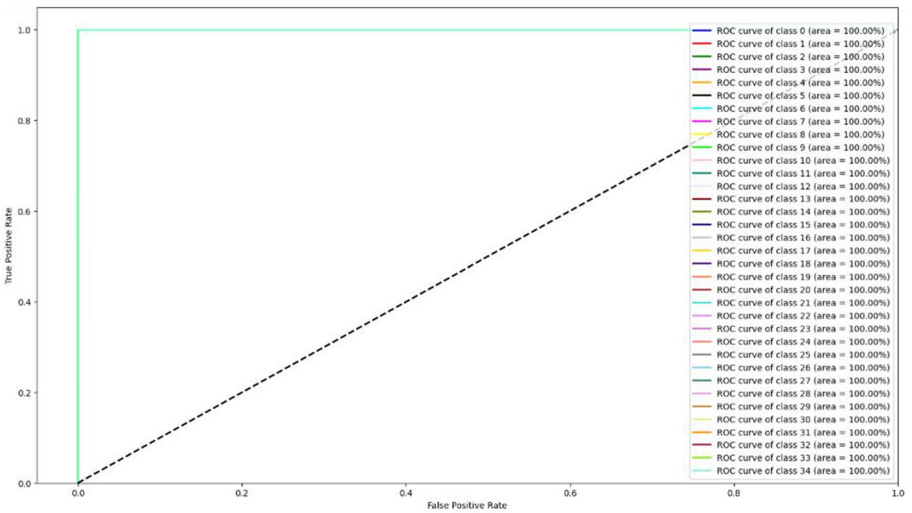

Fig. 11 ROC curve for CNN with data preprocessing

Table 4 Accuracy comparison for dataset2 with and without preprocessing

Dataset—2		
Classifiers	With preprocessing accuracy (%)	Without preprocessing accuracy (%)
RF	91.35	94.72
KNN	91.69	92.50
DT	88.80	72.43
ET	99.87	96.63
GNB	88.80	19.79
CNN	98.60	97.2

CNN achieved 99.99 with preprocessing whereas our ML model Decision Tree Classifier obtained 99.42%. Therefore, with preprocessing, our model is performing better for the dataset1 and the model performance comparison can be found in Fig. 9.

For the dataset2, ExtraTree Classifier is performing better than CNN with preprocessing.

We have used a graphical plot called the receiver operating characteristic (ROC) curve to show how our classifier system performs as the discrimination threshold is varied. At various classification thresholds, it shows the correlation between the true-positive rate (TPR) and the false-positive rate (FPR). Figure 10 is one such plot for KNN and Fig. 11 is for CNN model with preprocessing which achieved an accuracy of 99.99%. From Table 4, we can see that our GNB model without preprocessing gave

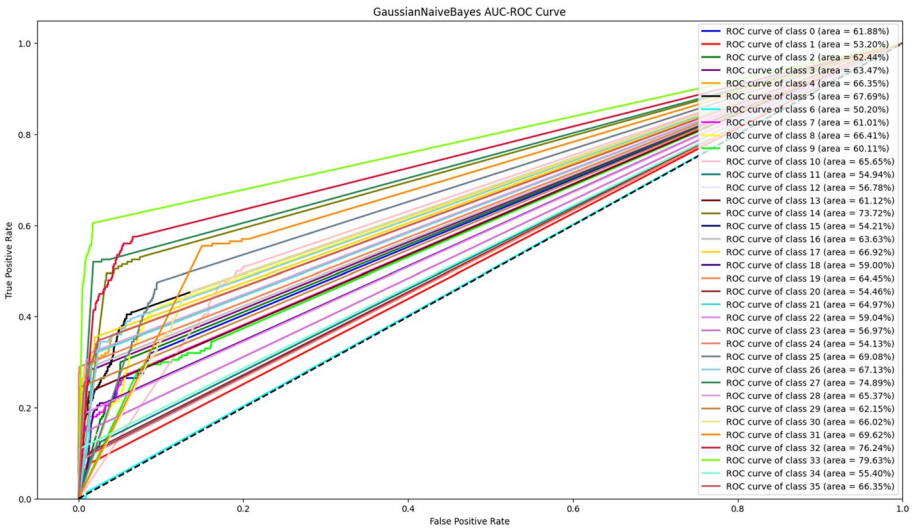

Fig. 12 ROC curve for GNB without preprocessing

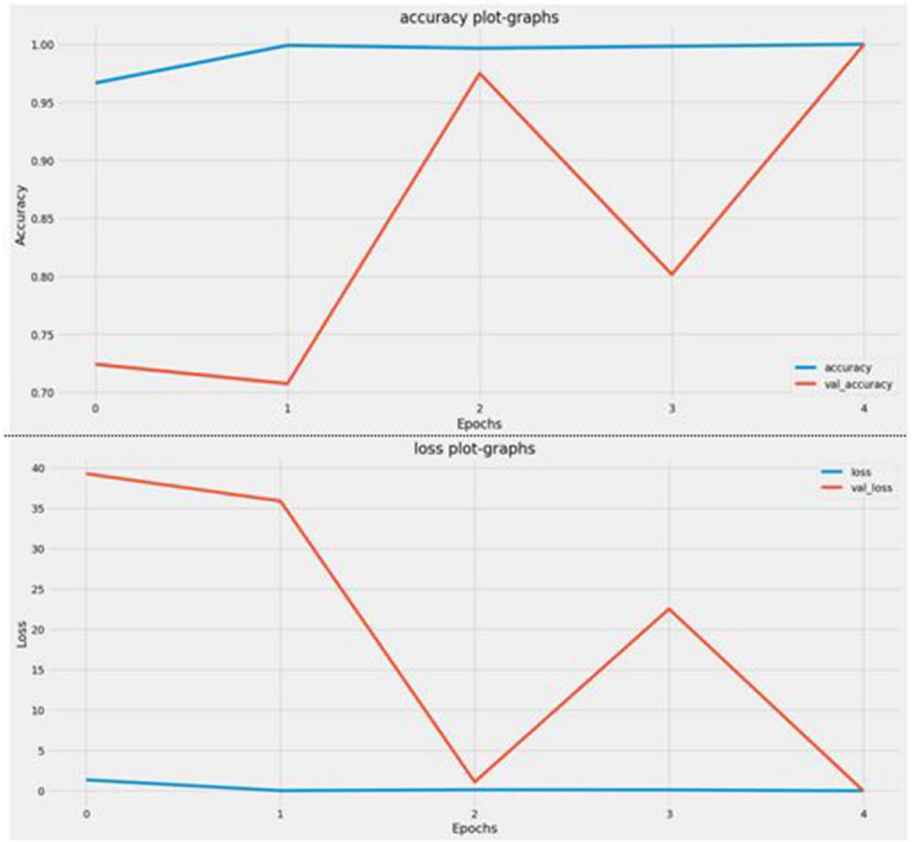

Fig. 13 Accuracy and loss plot for CNN

an accuracy of 19.79% and its ROC curve is as shown in Fig. 12 with so many variations in the curve as it achieved very less accuracy compared to others.

Accuracy Plot and Loss Plot

Precision: Precision is a performance metric that quantifies the ratio of correct positive predictions to the total number of positive predictions generated by the model:

$$\text{Precision} = \text{TP}/(\text{TP} + \text{FP}). \tag{6}$$

Recall: Recall is an evaluation metric that gauges the percentage of actual positive instances in a dataset that the model correctly identifies as positive:

$$\text{Recall} = \text{TP}/(\text{TP} + \text{FN}). \tag{7}$$

F1-Score: A measurement called the F1-Score is the harmonic mean of recall and precision:

$$F1 \text{ score} = 2 * (\text{Precision} * \text{Recall})/(\text{Precision} + \text{Recall}). \tag{8}$$

Accuracy: The percentage of the model's correct predictions among the total predictions made is measured by the accuracy metric. It indicates how accurate the model's predictions are:

$$\text{Accuracy} = (\text{TP} + \text{TN})/(\text{TP} + \text{TN} + \text{FP} + \text{FN}). \tag{9}$$

A true positive (TP) occurs when the model accurately predicts the positive class for an instance that is actually positive.

When the model accurately predicts the negative class for an instance that is actually negative, it is said to be a true negative (TN).

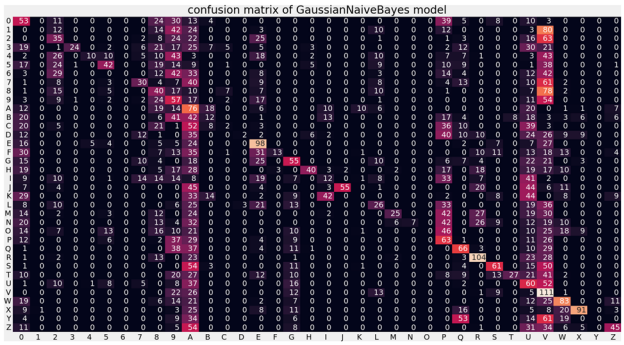

Fig. 14 Confusion matrix for GNB model

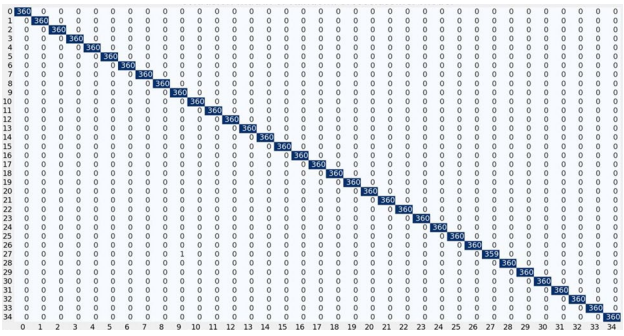

Fig. 15 Confusion matrix for CNN model

False positive (FP): In a false positive, the model incorrectly assigns a class to an instance that is really negative.

False negative (FN): A false negative is a type of error that occurs in a binary classification problem where the model incorrectly predicts a negative outcome for an instance that actually belongs to the positive class.

An accuracy plot is a graphical representation of the performance of a deep learning model during training. The plot

Table 5 Classification performance of CNN model with preprocess for dataset2

Class	Precision	Recall	F1-score
0	0.92	0.94	0.93
1	0.99	1.00	0.99
2	0.99	0.97	0.98
3	0.92	0.94	0.93
4	1.00	0.97	0.96
5	0.99	0.98	0.99
6	0.98	0.94	0.97
7	0.99	0.89	0.94
8	1.00	0.96	0.98
9	0.98	0.94	0.92
A	1.00	0.97	0.97
B	0.98	1.00	0.97
C	0.99	1.00	1.00
D	0.93	0.99	0.99
E	0.99	0.99	0.99
F	0.99	0.99	0.96
G	0.99	0.97	0.99
H	1.00	0.99	0.99
I	0.93	0.98	0.96
J	1.00	0.99	0.99
K	0.99	0.98	0.99
L	0.99	0.99	0.99
M	0.98	1.00	0.99
N	1.00	0.97	0.99
O	0.98	0.96	0.97
P	1.00	0.99	1.00
Q	0.99	0.93	0.96
R	0.99	0.99	0.99
S	0.99	1.00	0.99
T	0.99	0.99	0.99
U	1.00	1.00	0.99
V	1.00	1.00	1.00
W	0.85	1.00	0.99
X	1.00	0.99	0.99
Y	0.99	0.99	0.99
Z	1.00	1.00	1.00
Support	200		
Weighted AVG	98%		
Macro-AVG	98%		

in Fig. 13 shows the accuracy of the model on the training set and validation set as the model is trained over multiple epoch. Here we have used totally 4 epochs; it can be seen that the training accuracy line-plot and validation accuracy line-plot have grown steadily.

A loss plot is a graphical representation of the performance of a deep learning model during training. The plot displays the loss of the model on the training set and validation set as the model is trained over multiple epochs as in Fig. 13.

Figures 14 and 15 show the confusion matrix for GNB and CNN model.

The precision value of 100% achieved for classes A, U, P, and J indicate that all instances belonging to those classes were correctly identified by the model. The precision value of 93% achieved for class I indicates that 93% of instances belonging to that class were correctly identified, while the remaining 7% were misclassified as shown in Table 5.

Real-Time Recognition Using YOLO-NAS Model with Roboflow Dataset of ISL

YOLO-NAS is a next-generation object detection model that has been developed using the Neural Architecture Search (NAS) technology. NAS is an automated process that searches for the optimal neural network architecture for a particular task. It does this by exploring a vast search space of possible architectures and selecting the most efficient and high-performing one.

We have used this model as it provides us with an enhanced detection of objects, improved localization accuracy, and higher performance per-compute ratio. In addition, another reason for using this YOLO-NAS model is that it outperforms YOLO-v5, YOLO-v7 and YOLO-v8 models.

This model is pre-trained on Top Datasets such as COCO, Object365 and Roboflow100. YOLO-NAS is around 0.5 mAP (mean Average Precision point) point more accurate and 10–20% faster than other variants of YOLO. In specific, we are using YOLO-NAS-S (S—small) for our real-time experimentation.

Steps involved:

1. Exporting the dataset from Roboflow into colab Notebook.
2. Indian sign language dataset is an object detection dataset of each ISL letters with a bounding box as shown below.

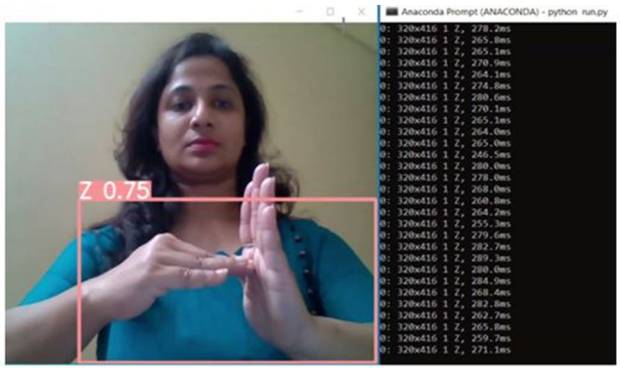

Fig. 16 YOLO-NAS-S model correctly predicting the gesture as *Z* with 0.75 confidence

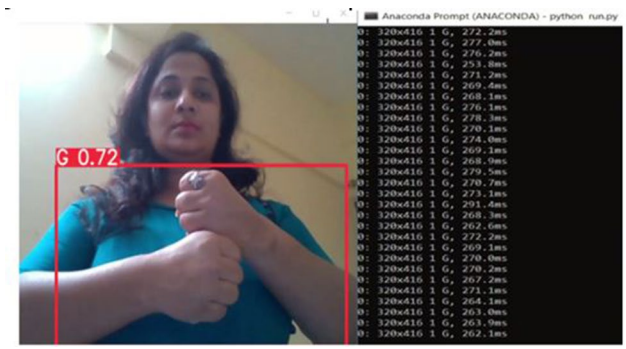

Fig. 17 Model correct prediction for the label G with 0.72 confidence

3. We have fine-tuned or trained our YOLO-NAS-S on this dataset. So that our model will read letters of this Indian sign language.
4. We have 9991 images in the dataset. In that, we used 1.5k for validation, 1.3k for testing and 7.2k for training set.
5. A health check is performed to test whether the dataset is balanced or not.
6. Upon extracting the dataset into the colab, we will have data.yaml file that contains the total number of classes and names information.

7. Once all the information is gathered and set, we will instantiate the YOLO-NAS-S model. We have set the pre-trained weights to COCO while instantiation.
8. A few arguments that we defined for training params are:
   Optimizer – Adam.
   max_epochs – 300.
   Loss – PPYoloELoss.
   learning_mode – cosine.
   initial_lr(learning rate) – 5e− 4.
9. After training the model on ISL Dataset, we will test whether our model reads the gestures or not.
10. Fetching the best model weight after the training is completed.
11. Now, we have evaluated our model on the test dataset.
12. Predict the best model; here, we can test by passing an image from validation set to check whether model predict gestures or not. We can also test by passing a recorded hand gesture video to the model.
13. But in our experimentation, we have tested it with live webcam feed.
14. Samples results with the correct predictions are shown in Figs. 16 and 17.

We can also try and experiment the same model performance on the custom dataset too. But here we need to capture the signs and prepare a custom dataset. Then, using annotation tool labelImg, we need to annotate the images having only hand portions and provide the labels for each image in the dataset. This creates a.txt file having information about the bounding box points. We have manually split our dataset to train and test for all the classes A to Z and 0 to 9 digits.

Once these folders are ready, we can train and test our model by following the same procedures as described in the steps 8 to 13. Figure 18 shows the sample data collection process to create custom dataset (Figs. 19, 20).

Fig. 18 Custom dataset collection

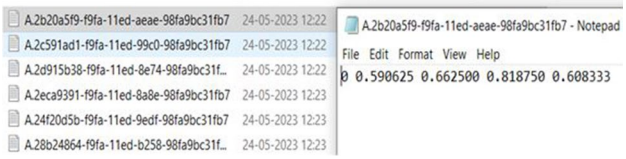

Fig. 19 labelImg—bounding box information for the cropped hand portion

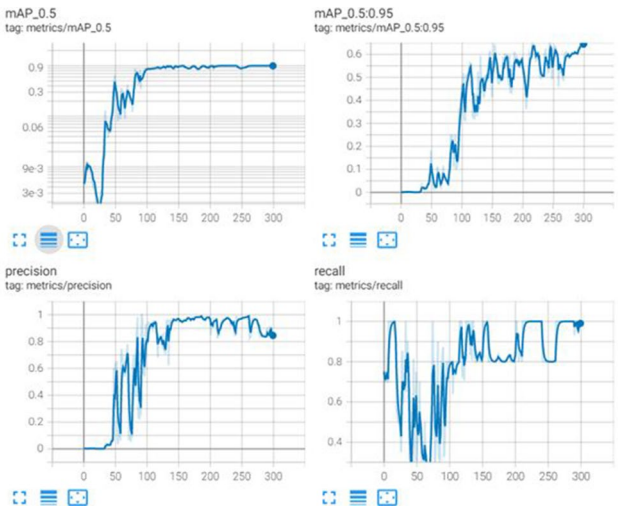

Fig. 20 Precision recall metrics graph

Comparison with Existing Systems

Table 6 presents a comparison between our proposed approach for recognizing Indian sign language and several other approaches that use publicly available datasets. Previous systems utilized machine learning-based classification algorithms, whereas our methodology is based on a CNN technique using deep learning. Remarkably, our proposed system achieved superior performance compared to all existing ISL systems, achieving an accuracy of 99.99% with the use of preprocessing methods and without data augmentation, but utilizing Adam as the optimizer.

Ensemble Probability Testing

Ensemble probability testing involves combining the predictions or probabilities from multiple models in an ensemble and utilizing them for making decisions or evaluating the uncertainty of the predictions as in Tables 7 and 8. We have used RF, KNN, and DT for the ensemble approach for dataset1.

Bias or large variation in individual models might result in under fitting or overfitting. Ensemble approaches can lessen these problems and produce a more balanced

Table 6 Proposed ISL system analysis in comparison to existing approaches

Author	Methods	Recognition (%)
Rahaman et al. [17]	K-Nearest Neighbors	95.95
Chowdhury and Uddin [18]	Support Vector Machine	97.90
Kishore and Rao [19]	Artificial Neural Network	98
Proposed system	CNN—without any augmentation, with preprocessing techniques and using 'Adam' optimizer	99.99
	ExtraTree Classifier for dataset 2 with and without preprocessing gained a very good accuracy	99.87 and 96.63, respectively
	Decision Tree with preprocessing	99.42
	Gaussian Naïve Bayes without preprocessing	95.47

prediction by integrating many models with differing biases and variances. Hence, we achieved higher accuracy than individual models; we can justify the same from Tables 3 and 4.

Table 7 Ensemble probability testing for dataset1

Dataset1—ensemble probability testing		
Classifiers	With preprocessing accuracy (%)	Without preprocessing accuracy (%)
RF	96.08	94.02
KNN		
DT		

Table 8 Ensemble probability testing for dataset2

Dataset2—ensemble probability testing		
Classifiers	With preprocessing accuracy (%)	Without preprocessing accuracy (%)
RF	93.08	79.02
KNN		
GNB		

Conclusion

An Indian sign language recognition system for English alphabet letters and numerals 0–9 has been developed. The system was trained using a large dataset of approximately 42,000 images for ML approach and 36,000 images for DL approach, which was processed using Google Colab. The system was created through the implementation of deep learning with convolutional neural network and five machine learning models. The system achieved an accuracy of 99.99% with the CNN and all five machine learning models had accuracy above 90%, with the ExtraTree model reaching 99.8%.

It has been observed that preprocessing has led to good results for machine learning (ML) methods, and deep learning (DL) methods when tested two datasets with diverse backgrounds and illumination variations present in the images of the datasets. In addition, ensemble probability testing performed the best than individual models. We also succeeded in recognizing the gestures in real time using YOLO-NAS-S Model for both custom dataset and Roboflow ISL dataset.

In fact, studying Indian sign language (ISL) using two datasets—one preprocessed and the other without can help other researchers better understand how machine learning (ML) and deep learning (DL) models behave in various circumstances. To evaluate the impact of preprocessing and without preprocessing, researchers can compare the performance metrics of ML and DL models trained on both datasets having controlled and uncontrolled environments in each.

In the future, word recognition using video inputs could be incorporated into the system, as well as the development of a model for generating complete sentence speech based on actions, which would be beneficial for deaf and mute individuals.

Acknowledgements The PES University in Bangalore, Karnataka, India provided the facilities needed to conduct the research, which the authors gratefully acknowledge.

Author Contributions PK selected the research issues, carried out the study, wrote the article, and examined the simulation findings with the supervision and support of SBJ.

Funding No funding received for this research.

Data Availability The data that support the findings of this study are openly available in the following link: dataset1: https://www.kaggle.com/datasets/vaishnaviasonawane/indian-sign-language-dataset, dataset2: https://www.kaggle.com/datasets/atharvadumbre/indian-sign-language-islrtc-referred, Roboflow ISL: https://universe.roboflow.com/isr1/indian-sign-language-3e2qh/dataset/4/images.

Declarations

Conflict of Interest No conflict of interest exists.

References

1. Deora D, Bajaj N. Indian sign language recognition, 2012 1st International Conference on Emerging Technology Trends in Electronics, Communication and Networking, IEEE 2012-978–1-4673-1627-9/12.
2. Nair AV, Bindu V. A review on indian sign language recognition. Int J Comput Appl. 2013;73(22):33–8.
3. Badenas J, Miguel Sanchiz J, Filiberto P. Motion-based segmentation and region tracking in image sequences. Pattern Recognit. 2001;34:661–70.
4. Liao P-S, Chen T-S, Chung P-C. A fast algorithm for multilevel thresholding. J Inf Sci Eng. 2001;17:713–27.
5. McIvor A, Zang B, Klette R. The background subtraction problem for video surveillance systems. In: International workshop robot vision 2001, Auckland, New Zealand, February 2001. Springer lecture notes in computer science. 1998. pp 176–83.
6. Sultana A, Rajapushpa T. Vision based gesture recognition for alphabetical hand gestures using the SVM classifier. Int J Comput Sci Eng Technol. 2012;3(7):218–23.
7. Nanivadekar PA, Kulkarni V. Indian sign language recognition: database creation, hand tracking and segmentation, International Conference on Circuits, Systems, Communication and Information Technology Applications, IEEE 2014, 978-1-4799-2494-3/14.
8. Raheja JL, Mishra A, Chaudhary A. Indian Sign language recognition using SVM. Pattern Recognit Image Anal. 2016;26(2):434–41.
9. Loke P, Paranjpe J, Bhabal S, Kanere K. Indian sign language converter system using an android app, International Conference on Electronics, Communication and Aerospace Technology, 2017 IEEE, 978-105090-5686-6/17.
10. Beena MV, Agnisarman Namboodiri MN. ASL numerals recognition from depth maps using artificial neural networks. Middle-East J Sci Res. 2017;25(7):1407–13.
11. Rodriguez KO, Chavez GC. Finger spelling recognition from RGB-D information using kernel descriptor, 2013 XXVI Conference on Graphics, Patterns and Images.
12. Molchanov P, Yang X, Gupta S, Kim K, Tyree S, Kautz J. Online detection and classification of dynamic hand gestures with recurrent 3D convolutional neural networks, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
13. Karpathy A, Toderici G, Shetty S, Leung T, Sukthankar R, Fei-Fei L. Large-scale video classification with convolutional neural networks. In CVPR, 2014.
14. Kpkl O, Gunduz A, Kose N, Rigoll G. Real-time hand gesture detection and classification using convolutional neural networks, paper accepted to IEEE International Conference on Automatic Face and Gesture Recognition (FG 2019).
15. Chen X, Gao K. Dense image network: Video spatial-temporal evolution encoding and understanding, paper submitted to ArXiv on 19 May 2018.
16. Kumar Makwana M. Sign language recognition, M. Tech thesis submitted to Indian Institute of Science, Bengaluru, June 2017.
17. Buehler P, Zisserman A, Everingham M. Learning sign language by watching tv (using weakly aligned subtitles). In: 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp 2961–2968. IEEE, 2009.
18. Cooper H, Ong E-J, Pugeault N, Bowden R. Sign language recognition using sub-units. J Mach Learn Res. 2012;13(1):2205–31.
19. Tharwat A, Gaber T, Ella Hassanien A, Shahin MK, Refaat B. Sift-based arabic sign language recognition system. In: Afro-European conference for industrial advancement. Cham: Springer; 2015. p. 359–70.
20. Quan Yang. Chinese sign language recognition based on video sequence appearance modeling. In 2010 5th IEEE Conference on Industrial Electronics and Applications, pages 1537–1542. IEEE, 2010.
21. Al-Rousan M, Assaleh K, Tala'a A. Video-based signer-independent Arabic sign language recognition using hidden markov models. Appl Soft Comput. 2009;9(3):990–9.
22. Badhe PC, Kulkarni V. Indian sign language translator using gesture recognition algorithm. In: 2015 IEEE International Conference on Computer Graphics, Vision and Information Security (CGVIS), pp 195–200. IEEE, 2015.
23. Starner T, Weaver J, Pentland A. Real-time American sign language recognition using desk and wearable computer based video. IEEE Trans Pattern Anal Mach Intell. 1998;20(12):1371–5.
24. Evangelidis GD, Singh G, Horaud R. Continuous gesture recognition from articulated poses. In: European conference on computer vision. Cham: Springer; 2014. p. 595–607.
25. Zhang J, Zhou W, Xie C, Pu J, Li H. Chinese sign language recognition with adaptive hmm. In: 2016 IEEE International Conference on Multimedia and Expo (ICME), IEEE, pp 1–6. IEEE, 2016.
26. Wang SB, Quattoni A, Morency LP, Demirdjian D, Darrell T. Hidden conditional random fields for gesture recognition. In: 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), vol. 2, pp 1521–1527. IEEE, 2006.
27. Sakoe H, Chiba S. Dynamic programming algorithm optimization for spoken word recognition. IEEE Trans Acoust Speech Signal Process. 1978;26(1):43–9.
28. Lichtenauer JF, Hendriks EA, Reinders MJT. Sign language recognition by combining statistical dtw and independent classification. IEEE Trans Pattern Anal Mach Intell. 2008;30(11):2040–6.
29. Nagarajan S, Subashini TS. Static hand gesture recognition for sign language alphabets using edge oriented histogram and multi class svm. Int J Comput Appl. 2013;82(4):28–35.
30. Zafrulla Z, Brashear H, Starner T, Hamilton H, Presti P. American sign language recognition with the kinect. In: Proceedings of the 13th International Conference on Multimodal Interfaces, pp 279–286, 2011.
31. Ming Lim K, Tan AWC, Chiang Tan S. Block-based histogram of optical flow for isolated sign language recognition. J Vis Commun Image Represent. 2016;40:538–45.
32. Kulkarni VS, Lokhande SD. Appearance based recognition of american sign language using gesture segmentation. Int J Comput Sci Eng. 2010;2(3):560–5.
33. Donahue J, Anne Hendricks L, Guadarrama S, Rohrbach M, Venugopalan S, Saenko K, Darrell T. Long-term recurrent convolutional networks for visual recognition and description. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 2625–2634, 2015.
34. Feichtenhofer C, Pinz A, Zisserman A. Convolutional two-stream network fusion for video action recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 1933–1941, 2016.
35. Camgoz NC, Hadfield S, Koller O, Bowden R. Using convolutional 3d neural networks for user-independent continuous gesture recognition. In: 2016 23rd International Conference on Pattern Recognition (ICPR), pp 49–54. IEEE, 20156.
36. Rokade YI, Jadav PM. Indian sign language recognition system. Int J Eng Technol. 2017;9(3):189–96.

37. Katoch S, Singh V, Shanker Tiwary U. Indian sign language recognition system using SURF with SVM and CNN. Array. 2022;14:100141.
38. Shenoy K, Dastane T, Rao V, Vyavaharkar D. Real-time Indian sign language (ISL) recognition. In: 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), pp 1–9. IEEE, 2018.