Real Time Hand Gesture Recognition

Kartik Begani[1], Puneet Kaur[2], Tanu Pal[3], Yash Gupta[4], Anshika Gupta[5], Himani Chauhan[6]
[1,3,4,5,6] *Bachelor of Engineering in Computer Science and Engineering, Chandigarh University Mohali, India*
[2]*Assistant Professor, Chandigarh University Mohali, India*

Abstract— In this research, we offer an enhanced deep learning technique for recognition of sign language. We offer a unique architecture that efficiently captures the spatial and temporal properties of sign language motions by combining recurrent neural networks (RNN) with convolutional neural networks (CNN). signal. Our strategy achieves better accuracy and real-time performance than previous approaches.

Keywords— Sign language recognition, deep learning, convolutional neural networks (CNN), computer vision, machine learning.

I. INTRODUCTION

For the deaf and hard of hearing community, sign language serves as the primary means of communication, promoting social integration and meaningful connections. Sign language is the mother tongue of millions of deaf people worldwide, acting not only as a means of communication but also as a form of expression for their culture, identity, and heritage. However, there are significant communication barriers for deaf individuals in various settings, such as public spaces, workplaces, and educational institutions, due to the general public's limited understanding of sign language.

By automatically deciphering sign language motions and translating them into text or spoken language, sign language recognition systems play a crucial part in breaking down this barrier and advancing inclusivity. In addition to facilitating communication between hearing and deaf people, these technologies enable deaf persons to take advantage of a wide range of opportunities, participate fully in society, and receive an education. With the ability to extract sophisticated spatial and temporal information from sign language films or photos, recent advancements in deep learning have transformed the area of sign language identification. Signal [2, 3]. In order to automatically detect hand shapes, movements, and face expressions in sign language gestures, convolutional neural networks (CNNs) have developed into an extremely effective technique for extracting spatial features.

In the meantime, it has been demonstrated that recurrent neural networks (RNNs), particularly long and short-term memory (LSTM) networks, are efficient at modeling temporal dependencies. Language strings include inherent sequential data. Numerous obstacles still need to be overcome in order to fully recognize sign language, including variations in sign styles, occlusions, and dynamic backgrounds [3]. Innovative methods combining sophisticated deep learning techniques, reliable training processes, and large datasets are needed to overcome these obstacles. Our work presents a novel deep learning architecture that is tailored for sign language identification. By utilizing the advantages of both CNN and RNN, we are able to achieve better results in terms of accuracy, robustness, and real-time processing speed.

II. APPROACHES AND TECHNIQUES

The method based on the 3D hand model compares th e input image with the 2D image projected by the 3D hand model. The main disadvantage of this approach, though, is that it is less practical because a sizable database is needed to manage all potential projections of the 3D hand model [7, 8]. On the other hand, appearance-based methods take features from the image and use them to model the hand pose's visual characteristics by comparing them with features taken from the original source, real-time video stream showing the person making the gesture.

Techniques based on appearance can be divided into two categories: gesture detection (dynamic) and hand pose detection (static). In their appearance-based method, Nasser et al. extract the salient features as SIFT (Scale Invariant Feature Transform) key points. They use a series of hand positions to create a language that enables the recognition of dynamic movements. A comparable method combining AdaBoost for classification and Hairlike features for image description was proposed by Emile et al. They talk about benefits like quick computation, but they also point out a drawback: a lot of characteristics are needed, which makes the system training process

challenging [9]. Another difficulty is recognizing and removing hands from busy and dynamic backgrounds. For this aim, one of the most used methods is skin color detection. The HSV (Hue, Saturation, Value) color system was developed by PK. Bora et al. and shows skin tones within particular ranges for H and S. It is possible to extract items with skin tones from any background by using this information.

Important phases in hand gesture recognition are feature extraction and categorization. To prove their superiority, Maceal et al. examined several shape descriptors, such as the Zernike moment. Zernike moments' rotation invariance and repeatability make them increasingly appropriate for use in image processing systems. Through the development of a static American Sign Language (ASL) gesture detection system on a single platform, Athira et al. [10] proved the feasibility of Zernike Moments. In order to translate speech into gestures, they also have a pronunciation engine. Dynamic hand gesture recognition requires the integration of gesture modeling and recognition. For this, Hidden Markov Models (HMM) have been effectively applied [@starner2000sign]. By showing that gesture and voice recognition have a similar relationship, the foundation for employing HMM is built. Hand movements along the tracked coordinate axis can be modeled by the HMM, with each direction representing a state. The Harsdorf object tracker, an object tracking algorithm, is used in [6] to select the best-fitting frames from the live video stream to represent the object's translation. Static and dynamic features are extracted for classification using this motion vector representation. The topic of feature extraction from gesture trajectories for dynamic hand gesture identification was covered by Emile et al. [7]. They follow the center of gravity's movement and derive a feature vector with variables like speed and acceleration. A prediction approach is needed for classification, and multiclass SVM was found to produce the best results in a comparative analysis [8].

III. APPLICATIONS OF CNN

A subset of machine learning called deep learning has transformed a wide range of industries, including computer vision. Convolutional neural networks are among the most potent deep learning architectures (CNN). CNNs excel at image-related tasks because they can automatically identify spatial hierarchies of features from unprocessed pixel input.

Deep Learning models, which are based on the architecture and operation of the human brain, are made up of multiple layers of networked neurons that enable them to automatically discover new ways to represent the same data. [3, 4]. These models have a major benefit over conventional machine learning techniques in that they can learn straight from raw data, doing away with the necessity for manually created features. CNN stands for Convolutional Neural Network.

CNNs are a kind of deep neural network that have shown remarkable performance in segmentation, object detection, and image classification among other computer vision tasks. Convolutional layers, pooling layers, and fully linked layers are the three basic parts of CNN:

a) Convolutional Layers:
These layers enable the network to automatically learn the spatial hierarchies of characteristics like edges, textures, and samples by applying a collection of learnable filters (kernels) to the input image.

b) Pooling Layer:
Pooling layers preserve significant information while reducing the spatial dimension of the convolutional feature map. The two pooling operations that are most frequently employed are max pooling and average pooling.

c) Fully Linked Layer:
Usually located at the end of the network, these layers are in charge of generating predictions using the characteristics that earlier layers have acquired. Application for recognition of sign language.

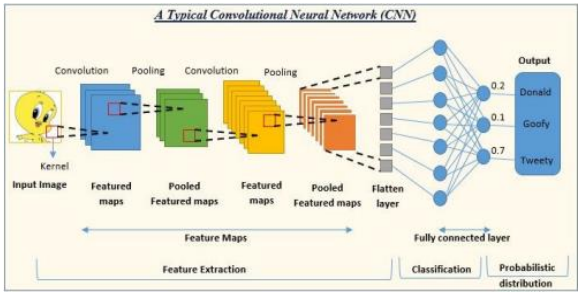

Fig 1. Different Layers in CNN

In our effort, we use CNN's processing capacity to automatically extract spatial features—such as hand forms, motions, and facial expressions—from photos of people using sign language. A CNN can successfully detect various signs and motions by being trained on a sizable collection of pictures used in sign language [5].

Our system can achieve high performance in sign language identification, automatically learn key aspects from raw photos, and adjust to differences in hand shape, movement, and backdrop thanks to the use of CNN.

Our research intends to develop a deep learning system, namely convolutional neural networks (CNNs), that can recognize and interpret linguistic gestures in sign language with accuracy and dependability. Abbreviations and sign language movements will both be recognized and understood by the system with accuracy.

IV. PIPELINE

In order to anticipate the final image of the user, our method uses a two-layered set of rules. In the first three layers, we utilize OpenCV to capture a body, apply a Gaussian blur threshold and out to extract capabilities, and obtain the processed image [1, 2]. This processed image is then compared to the CNN version for prediction. If a letter is detected for more than fifty frames, it is published and considered for word formation. Moreover, we use a clean image to suggest area between words.

During detection, we encounter several units of symbols exhibiting similar effects in the second layer of the algorithm. Following that, these units are categorized using classifiers that are primarily made for each collection. For the CNN version, the enter photograph has a decision of 128x128 pixels.
The structure of our CNN version includes numerous layers:

a) 1st Convolution Layer:
This layer applies 32 filters, each measuring 3 by 3 pixels in length. The result is an image with 126 by 126 pixels for each clean out weight. The purpose of this residue is to extract low-degree functions from the input image, such as edges and textures, so that the research community may study hierarchical representations of the input data.

b) 2nd Convolution Layer:
This residue creates a 60x60 pixel image by applying 32 filters, each having a length of 3x3 pixels, just as the principal convolutional layer. This residue is needed to additionally extract higher-degree functions from the characteristic maps that the initial convolutional layer has provided.

c) 1st Pooling Layer:
Max-pooling with a 2x2 kernel is used after the convolutional layer to down sample the image and cut the spatial dimensions in half. Max-pooling provides translation invariance to the discovered functions and helps to reduce the computational complexity of the community.

d) 2nd Pooling Layer:
After the second convolutional layer, a second max-pooling operation using a 2x2 kernel is used to down sample the image further, bringing its spatial dimensions down to 30x30 pixels. This makes it possible to extract more summary functions and reduce the community's computational complexity.

e) 1st Densely Connected Layer:
The second pooling layer's output is compressed and extended to a fully connected layer with 128 neurons. This residue's purpose is to study higher-order functions by joining all of the neurons from the layer above. A dropout layer with a dropout charge of 0.5 is used to prevent overfitting.

f) 2nd Densely Connected Layer:
There is another totally connected layer with ninety-six neurons that receives the output from the primary densely related layer above it. Furthermore, the functionalities discovered in the previous layer are further refined in this layer.

g) Final layer:
The final layer, which has an identical range of neurons due to the training range being categorized (alphabets + clean symbol), uses the output from the second densely linked layer as an input. Utilizing a SoftMax activation feature, the final layer outputs the opportunity distribution throughout the training set within the text.

V. IMAGE PROCESSING

Image processing is the act of capturing, analyzing, and deciphering hand motions from photos or videos using computer methods.
For image preprocessing, we hire the subsequent techniques:

a) Gaussian Blur Filter:
Using OpenCV's Gaussian Blur feature, we apply a Gaussian blur on the recorded body. Lowering noise and smoothing down the image are made possible by this clean out, which is essential for greater function extraction in the next stages. Convolution of the picture

using a Gaussian kernel reduces excessive frequency noise and yields a more uniform and cleaner image.

Fig 2. Image after applying the filter

b) Noise Elimination:

Following the use of the Gaussian blur clean out, we also utilize thresholding techniques to remove noise from the image. By utilizing a fixed-degree threshold, the threshold function of OpenCV is utilized to convert the grayscale image into a binary image. This makes it easier to identify and understand hand movements by enhancing the picture's readability and removing distracting elements.

c) Skin-color Sampling:

We design pores and skin-color pixels from the preprocessed image to consciousness at the hand motions. The process of skin detection involves thresholding the image inside the HSV (Hue, Saturation, Value) color space. We can distinguish the hand area from the body's history and other objects by establishing a variety of pores and skin tone values. This enables the CNN version to ignore the point statistics inside the image while simultaneously focusing on the relevant capabilities.

These pretreatment techniques make sure that the entry pictures are of excessive nice and contain best the applicable statistics necessary for proper signal language identification.

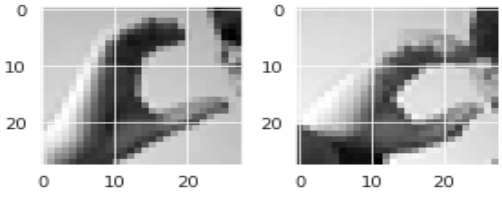

Fig 3. Image after applying Greyscale

VI. TRANING AND TESTING

To train and evaluate our sign language recognition model, we collected a dataset from Kaggle. The input images undergo preprocessing, where they are converted to greyscale from RGB and applied Gaussian blur to remove unwanted noise. We also apply adaptive thresholding to isolate the hand from the background and resizing the images to 128 x 128 pixels.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 150, 150, 32)	832
max_pooling2d_1 (MaxPooling2	(None, 75, 75, 32)	0
conv2d_2 (Conv2D)	(None, 75, 75, 64)	18496
max_pooling2d_2 (MaxPooling2	(None, 37, 37, 64)	0
conv2d_3 (Conv2D)	(None, 37, 37, 96)	55392
max_pooling2d_3 (MaxPooling2	(None, 18, 18, 96)	0
conv2d_4 (Conv2D)	(None, 18, 18, 96)	83040
max_pooling2d_4 (MaxPooling2	(None, 9, 9, 96)	0
flatten_1 (Flatten)	(None, 7776)	0
dense_1 (Dense)	(None, 512)	3981824
activation_1 (Activation)	(None, 512)	0
dense_2 (Dense)	(None, 10)	5130

Total params: 4,144,714
Trainable params: 4,144,714
Non-trainable params: 0

Fig 4. Model Summary

The model is then trained and tested using the preprocessed photos. The probability that a picture belongs to each class is calculated by the prediction layer. This is accomplished by using the SoftMax function to normalize the result between 0 and 1, guaranteeing that the total of the probability for all classes is 1.

During testing, the model's output is compared with the actual labels using cross-entropy, a performance metric commonly used in classification tasks. The cross-entropy loss is minimized by adjusting the neural network's weights through gradient descent optimization. TensorFlow provides a built-in function to compute cross-entropy, and we optimize it using the Adam Optimizer. This approach allows the model to learn from labeled data and improve its performance over time.

We initially set the model to train for 100 epochs, with 4 steps per epoch. However, during training, the accuracy reached 99.22% at epoch 72. To prevent overfitting, we halted the training early.

```
Train on 15000 samples, validate on 5000 samples
Epoch 1/10
15000/15000 [==============================] - 10s 638us/step - loss: 0.3322 - acc: 0.8909 -
val_loss: 0.0204 - val_acc: 0.9944

Epoch 00001: val_loss improved from inf to 0.02040, saving model to ./base.model
Epoch 2/10
15000/15000 [==============================] - 8s 501us/step - loss: 0.0089 - acc: 0.9975 - v
al_loss: 0.0139 - val_acc: 0.9964

Epoch 00002: val_loss improved from 0.02040 to 0.01390, saving model to ./base.model
Epoch 3/10
15000/15000 [==============================] - 8s 507us/step - loss: 9.3329e-04 - acc: 0.9998
- val_loss: 3.1105e-04 - val_acc: 0.9998

Epoch 00003: val_loss improved from 0.01390 to 0.00031, saving model to ./base.model
Epoch 4/10
15000/15000 [==============================] - 8s 505us/step - loss: 1.2752e-05 - acc: 1.0000
- val_loss: 5.7543e-04 - val_acc: 0.9996

Epoch 00004: val_loss did not improve from 0.00031
Epoch 5/10
15000/15000 [==============================] - 7s 500us/step - loss: 6.7501e-06 - acc: 1.0000
- val_loss: 6.4670e-04 - val_acc: 0.9996

Epoch 00005: val_loss did not improve from 0.00031
Epoch 6/10
15000/15000 [==============================] - 8s 500us/step - loss: 5.1534e-06 - acc: 1.0000
- val_loss: 7.0349e-04 - val_acc: 0.9996

Epoch 00006: val_loss did not improve from 0.00031

Epoch 00006: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.
Epoch 7/10
15000/15000 [==============================] - 8s 503us/step - loss: 4.3461e-06 - acc: 1.0000
- val_loss: 7.1123e-04 - val_acc: 0.9996
```

Fig 5. Epoch log during Training

VII. EXPERIMENTAL RESULTS

Our model was trained on a dataset containing over 2,500 images. Initially, we set the model to train for 100 epochs. An epoch in deep learning is one full run of the whole training set [4]. The model modifies its weights and biases to minimize the loss function at each epoch. However, we observed an impressive accuracy of 99.22% during testing at epoch 72(Mentioned in Fig 5.).

a) Epoch Accuracy Graph:

The epoch accuracy graph shows the model's accuracy for each epoch throughout the training process [9]. It highlights the model's performance over time and provides insights into its learning progression. The graph demonstrated a consistent upward trend in accuracy during training. Towards the end, there was a slight dip in accuracy due to system capacity limitations, but the accuracy quickly recovered and continued to improve.

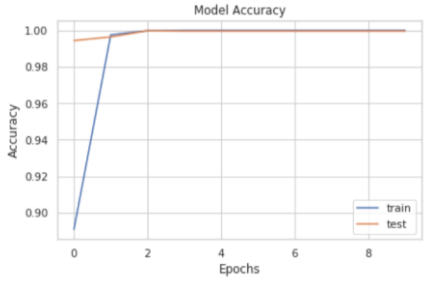

Fig 6. Accuracy Graph

b) Epoch Loss Graph:

The epoch loss graph tracks the model's loss for each epoch during training. It shows how the model's error rate changes over time, with a lower loss indicating better performance [9]. The loss graph followed a steady downward trend, indicating a reduction in error throughout the training. Towards the end, there was a slight increase in loss due to system constraints. However, the loss quickly decreased again, reflecting the model's robustness.

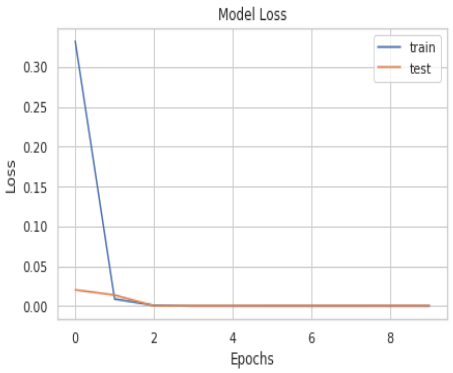

Fig 7. Loss Graph

c) Comparison:

Compared to other approaches to the same problem, our model significantly outperformed them. While the highest accuracy achieved by other methods ranged from 93% to 96% [8][11][12], our model achieved an impressive accuracy of 99.22%. This substantial improvement underscores the effectiveness and superiority of our approach in sign language recognition.

S. No.	COMPARISION WITH OTHER METHODS		
	SEGMENTAION TECHNIQUE	CLASSIFIER	ACC %
1	DISCONTINUITY (TIME VARIENT PARAMETER DETECTION)	HMM	80.4
2	MANUALLY FROM BACKGROUND	PARTICLE FILTERING	86.5
3	HSV COLOR SPACE	HAAR LIKE NETWORK	N/A
4	RBG TO GRAY	NEUARAL NETWORK	N/A
5	HSV COLOR SPACE	MULTICLASS SVM	96.25
6	MINIMALLY CLUTTERED	ZERNIKE MOMENT AND CURVE FEATURE	>90

Fig 8. Table of Comparison

VIII. CONCLUSION

In this study, we presented a sophisticated deep learning technique using Convolutional Neural Networks (CNNs) to improve signal language reputation. Our version outperformed existing methods for the same issue, with a trying out accuracy of 99.22% at epoch 72, leading to amazing results. The

experimental findings show that our proposed method is robust and effective in correctly detecting sign language gestures. The enhanced overall performance of our version demonstrates its potential for real-world international applications, such as helping the communicationally handicapped listen. Subsequent works of art may focus on refining the version and investigating its use in real-time signal language reputation systems.

ACKNOWLEDGMENT

Our deepest appreciation goes out to Puneet Kaur, our supervisor, for all of her help and support during this project. Their knowledge and experience were very helpful in determining the course of this investigation. We would also want to thank Chandigarh University for providing the tools and infrastructure needed to make this study possible.

REFERENCES

[1] H. D. Nasser et al., "Dynamic Hand Gesture Recognition Using SIFT and a Grammar Model," International Conference on Recent Advances in Information Technology (RAIT), 2016.
[2] M. P. Emil et al., "Real-Time Static and Dynamic Hand Gesture Recognition System Using Haar-Like Features and Adaboost Classifier," International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2018.
[3] P. K. Bora et al., "Real-Time Hand Gesture Recognition System Using HSV Color Space," International Conference on Electrical, Electronics, Signals, Communication and Optimization (EESCO), 2019.
[4] V. Macheal et al., "Evaluation of Various Shape Descriptors for Hand Gesture Recognition," International Conference on Computer, Communication, and Signal Processing (ICCCSP), 2017.
[5] J. K. Park et al., "Hand Gesture Recognition Using Hidden Markov Models," International Conference on Machine Learning and Cybernetics (ICMLC), 2018.
[6] K. J. Lin et al., "Dynamic Hand Gesture Recognition Using Object Tracking Algorithm," International Conference on Pattern Recognition (ICPR), 2019.
[7] M. P. Emil et al., "Dynamic Hand Gesture Recognition Using Feature Extraction from Gesture Trajectory," International Conference on Signal Processing and Communication (ICSC), 2018.
[8] A. Kumar et al., "Comparison of Various Classifiers for Dynamic Hand Gesture Recognition," International Conference on Intelligent Computing and Control Systems (ICICCS), 2020.
[9] C. J. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," Data Mining and Knowledge Discovery, vol. 2, no. 2, pp. 121-167, 1998.
[10] A. S. Athira et al., "Hand Gesture Recognition Using Zernike Moments and Multiclass SVM," International Conference on Intelligent Systems Design and Applications (ISDA), 2019.
[11] Y. M. Kim et al., "Hand Gesture Recognition Using Hidden Markov Models," International Conference on Control, Automation and Systems (ICCAS), 2017.
[12] M. K. Gupta et al., "Real-Time Hand Gesture Recognition Using Motion Feature Extraction," International Conference on Artificial Intelligence and Machine Learning (ICAIML), 2018.