International Journal of Innovative Research in Computer Science and Technology (IJIRCST)
ISSN (Online): 2347-5552, Volume-14, Issue-6, Nov-Dec 2026
DOI: https://doi.org/10.55524/ijircst.2026.14.6.7
Article ID IRP-1803, Pages 58-63
www.ijircst.org

Sign Language Alphabet Recognition and Conversion to Text Using MediaPipe and Machine Learning

Nitesh Kumar[1], Anmol Raj[2], Harsh Kumar Gautam[3], Aditya Raj[4], and *Yojna Arora[5]

[1, 2, 3, 4, 5] Department of Computer Science & Engineering, Sharda University, Greater Noida, Uttar Pradesh, India

*Correspondence should be addressed to Yojna Arora; yojana183@gmail.com

Received: 19 Oct 2026; Revised: 3 Nov 2026: Accepted: 16 Nov 2026; Published Online: 26 Nov 2026

Copyright © 2026 *Yojna Arora et. al. This is an open-access article distributed under the Creative Commons Attribution 4.0 International License (CC BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

ABSTRACT- Many deaf and hard-of-hearing people sign, yet very few hearing people can read signs, so an everyday conversation often ends up needing a written note or an interpreter. We built a small system that watches a camera, reads the American Sign Language (ASL) fingerspelling alphabet, and types the letter on screen. MediaPipe finds 21 hand landmarks in each frame, we turn them into 63 numbers that do not depend on where the hand sits in the picture, and a Random Forest chooses the letter. Our own recordings cover 24 static letters (J and Z need motion, so we left them out), about 400 samples each, split 80:20 into training and test data. On the held-out 20% the classifier reached 70% accuracy. In live use it was less steady than that number suggests, because neighboring video frames are almost identical and results change with lighting and camera position. The pipeline runs on a Windows laptop without a GPU. It recognizes single letters only, not words or sentences. We also describe where it fails and what we would change next.

KEYWORDS- ASL Fingerspelling, Hand Landmarks, MediaPipe, Random Forest, Sign Language Recognition

I. INTRODUCTION

Hearing people rarely think about how much of daily life depends on speech. A bank counter, a doctor's room, a classroom, and a bus stop all assume that both sides can talk to each other. For many deaf and hard-of-hearing people a sign language takes that place, and the World Health Organization (WHO) counts people with hearing loss in the hundreds of millions [1]. American Sign Language (ASL), like other sign languages, is a language in its own right, with its own grammar and vocabulary. It is not English spelled out on the hands [2]. The difficulty is that most hearing people never learn it. A deaf signer in a public place therefore ends up waiting for an interpreter who may not be there, or typing on a phone.

Automatic sign language recognition (SLR) tries to close this gap by letting a computer watch the signer and write down what it sees. That is harder than it sounds. A sign is more than a hand shape. Location, movement, palm orientation, and the signer's face and body all carry meaning [2], [3], and people differ in speed, style, and hand size. Real rooms add poor lighting, cluttered backgrounds, and fingers that hide behind each other. Recent surveys describe continuous signing in uncontrolled conditions as still largely unsolved [4]–[6].

A common way to make progress is to start small. ASL has a manual alphabet, and spelling a word letter by letter (fingerspelling) is how signers handle names, places, and technical terms that have no sign of their own [7]. Most of the letters are static hand shapes, so a single frame is enough to tell them apart. That makes the alphabet a fair first target for a system that must run on ordinary hardware. A working letter recognizer is also useful for people who are only beginning to learn to sign.

Earlier sign and fingerspelling systems often needed something extra: colored gloves, depth sensors, or large neural networks that want a graphics processing unit (GPU) [7]–[9]. MediaPipe Hands changed that for us. It finds 21 three-dimensional landmarks on a hand from a normal red-green-blue (RGB) camera, fast enough to run on a phone [10]. If the classifier is given those landmarks instead of raw pixels, most of the background and lighting variation never reaches it, and each frame shrinks to a short list of numbers. A Random Forest [11] can handle a vector like that on a laptop central processing unit (CPU) with no special hardware.

This paper describes the system we built along those lines. A camera frame goes in, MediaPipe returns the hand landmarks, we convert them into 63 numbers that do not depend on where the hand sits in the picture, and a Random Forest decides which letter is being shown. The letter is added to an on-screen text string once the prediction has stayed steady for a few frames. The task is therefore: given one frame with one hand showing a static ASL letter, return the right letter fast enough to type with it, and do not fill the text with stray letters while the hand moves between signs. The scope is narrow on purpose. We cover 24 static letters (A to Y without J and Z, which need movement), single letters only, and no words or sentences.

We make four contributions. First, we recorded our own landmark dataset for the 24 letters. Second, we show that MediaPipe plus a Random Forest is enough for a working demonstration on a laptop with no GPU. Third, we add a simple confidence-and-stability rule so that the output text does not flicker. Fourth, we report accuracy on a held-out test set and explain why that number is more optimistic than the live behavior. Section II reviews earlier work, Section III describes the design, Section IV covers the implementation and dataset, Section V gives the results, and

Section VI concludes.

II. RELATED WORK

We group earlier work by what it was trying to do: recognize gestures in general, spell out letters, translate signed sentences, and track hands. Table 1 lists the papers we found most useful.

A. Early gesture and sign recognition

The survey by Mitra and Acharya [12] splits gesture recognition into detecting the hand, extracting features, and classifying them, and points out how much cluttered backgrounds and user-to-user variation get in the way. One of the first real-time ASL systems, by Starner et al. [8], used hidden Markov models on video from a desk-mounted or a cap-mounted camera and handled sentences from a small vocabulary. Cooper et al. [3] later wrote an overview that explains how hand shape, position, motion, and non-manual cues are modeled, and why a system trained on some signers often fails on new ones.

B. Fingerspelling recognition

Fingerspelling has been treated as a problem of its own. Pugeault and Bowden [7] built a real-time ASL fingerspelling recognizer from color and depth images with a random-forest style classifier, and they released a set of fingerspelling images that others have used since. Bantupalli and Xie [9] paired a convolutional network with a recurrent one to recognize ASL from video. Both show that static letters can be handled with standard tools. They also show the weakness we ran into ourselves: accuracy drops when the signer, the background, or the camera changes.

C. Deep learning and translation

Pigou et al. [13] used convolutional networks to recognize gestures in video. Camgoz et al. [14] went further and treated sign language translation as a sequence-to-sequence problem that maps signing video to spoken-language sentences. When timing matters, recurrent models such as long short-term memory (LSTM) networks [15] are the usual choice. Koller [4], Rastgoo et al. [5], and Adaloglou et al. [6] review this work. A recurring theme is that reported accuracy depends heavily on the dataset and on how the test set is chosen. Large general gesture sets such as HaGRID [16] help research on hand gestures, although they are not ASL alphabet data.

D. Landmark-based hand tracking

MediaPipe Hands [10] first finds the palm and then regresses 21 landmarks from the cropped hand. It sits on the MediaPipe framework for perception pipelines [17], and its palm detector belongs to the same family of lightweight mobile detectors as BlazeFace [18]. Because the output is only 63 numbers per hand, classifying it is far cheaper than processing an image, and clothing and background matter less. Random Forests [11] as implemented in scikit-learn [19] suit that kind of table-like input.

E. Where our work fits

None of the papers above targets exactly what we built: a small landmark-based letter recognizer that a student can reproduce on a laptop. We do not claim to beat the deep models. Our aim is a simple system whose behavior we can explain, together with an honest account of the gap between test accuracy and live behavior.

Table 1: Literature survey of related work

Authors	Focus	Method / contribution	Relevance to this work
Starner et al. [8]	Real-time ASL from video	Hidden Markov models on hand features from a desk or cap camera; small vocabulary	Early proof that real-time ASL recognition is feasible
Mitra and Acharya [12]	Gesture recognition survey	Reviews methods and open problems	Frames our pipeline: detect, extract features, classify
Cooper et al. [3]	Sign language recognition overview	Covers hand shape, motion, location, and non-manual cues	Explains why static letters are an easier starting task
Pugeault and Bowden [7]	ASL fingerspelling	Color and depth images with a random-forest style classifier	Closest earlier task; supports tree ensembles for letters
Pigou et al. [13]	Sign recognition with convolutional networks	Convolutional networks applied to gesture video	Learning-based recognition works but needs more compute
Camgoz et al. [14]	Sign language translation	Sequence-to-sequence model from signing video to sentences	The long-term goal beyond single letters
Bantupalli and Xie [9]	ASL recognition from video	Convolutional features plus a recurrent model	Deep baseline; needs more data and compute
Koller [4]	Survey of sign language recognition	Reviews progress, data, and evaluation practice	Context for how results depend on the evaluation protocol
Zhang et al. [10]	On-device hand tracking	21 hand landmarks from an RGB camera in real time	Source of our input features
Rastgoo et al. [5]	Deep-learning survey	Methods, input types, and datasets	Places our lightweight approach in context
Adaloglou et al. [6]	Comparison of deep SLR methods	Several methods tested on several datasets	Context for how dataset and protocol affect reported accuracy

III. SYSTEM DESIGN

We built the system around one decision: seeing the hand and naming the letter are two separate jobs. MediaPipe does the first. A small classifier trained on our own recordings does the second. Keeping them apart made debugging easier. When the output was wrong, we could check whether the landmarks were bad or the classifier had made a mistake, and either part could be swapped without touching the other.

A. Overview

Figure 1 shows the data flow. A frame from the camera goes to MediaPipe, which returns landmarks if it finds a hand. The landmarks are normalized into a 63-value vector and passed to the Random Forest, which returns a letter and a probability. A stability check then decides whether that letter should be typed. If no hand is found, nothing is predicted and the check starts over.

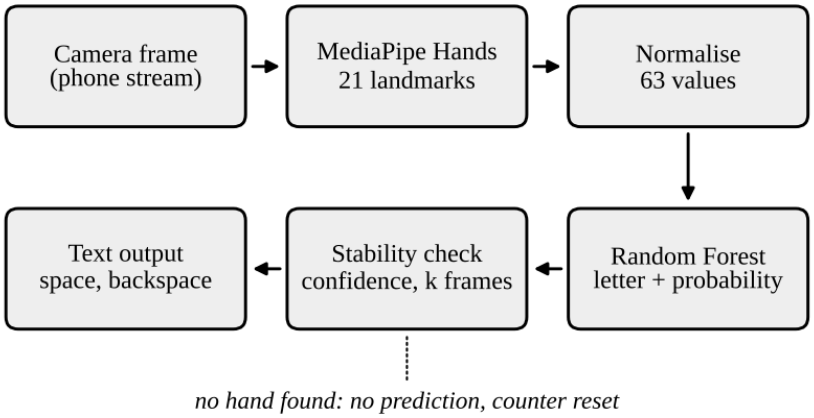


Figure 1: Data flow of the proposed system.

B. Modules

The design has seven modules, listed in Table 2. Each one does a single job and hands its result to the next.

Table 2: Modules of the system

No.	Module	Task
1	Input capture	Read frames from the camera (phone stream or webcam).
2	Landmark extraction	Detect the hand with MediaPipe and return its 21 landmarks.
3	Feature preparation	Build 63 values, taken relative to the wrist and scaled by hand size.
4	Classification	Predict the letter with the trained Random Forest.
5	Character mapping	Convert the predicted class label to its character.
6	Text output	Show the letter, build the text string, handle the space and backspace keys.
7	Evaluation	Compare predictions with labeled test samples and inspect the errors.

C. Feature representation

MediaPipe returns 21 landmarks for each detected hand, $\mathbf{p}_i = (x_i, y_i, z_i)$, where landmark 0 is the wrist and the others cover the thumb and the four fingers. Raw coordinates change when the hand moves across the frame or toward the camera, so they make a poor input. We therefore shift every point so that the wrist becomes the origin and divide by a hand-size value s computed from the landmarks themselves. The normalized points are

$$\mathbf{q}_i = (\mathbf{p}_i - \mathbf{p}_0) / s, \quad i = 0, 1, \ldots, 20 \qquad (1)$$

Stacking the 21 normalized points gives one feature vector $\mathbf{x}$ with 63 entries. This makes the features roughly independent of position and distance, so the same hand shape gives similar numbers whether the hand is near or far, left or right. It does not fix rotation of the hand, differences in hand proportions between people, or mistakes made by the landmark detector. Those remain sources of error.

D. Deciding when to type a letter

A classifier answers on every frame, including the frames in which the hand is moving from one sign to the next. If every answer were typed, the text would fill up with junk. The system therefore follows this loop: capture a frame, extract landmarks, normalize, classify, check confidence and stability, type the letter if both checks pass, and repeat. The price of the rule is delay. If k consecutive frames are required and the system runs at f frames per second, a letter appears about k/f seconds after the sign is held steady.

IV. IMPLEMENTATION

We wanted something that other students could rebuild: free software, a laptop with no dedicated graphics card, and a phone as the camera. This section lists the setup, describes how we recorded the data, and explains training and the live loop.

A. Hardware and Software

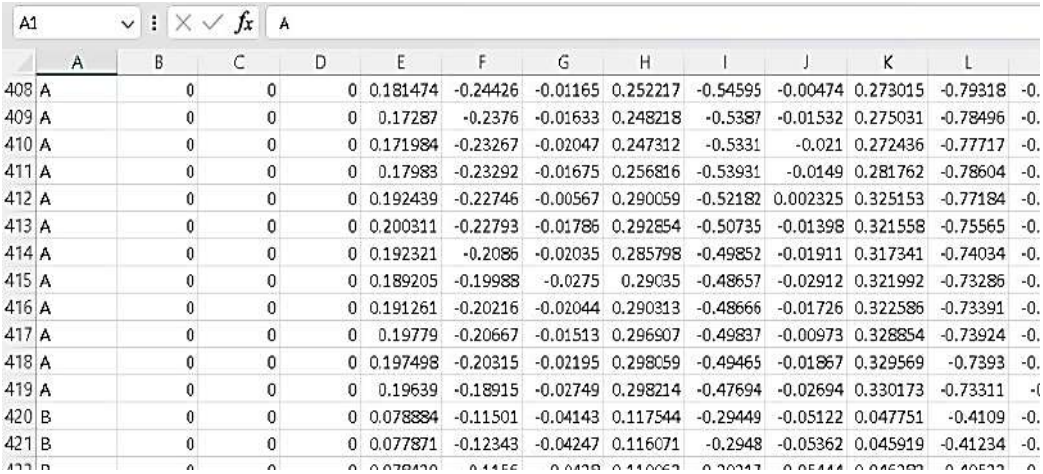
The code is written in Python 3.11. Hand landmarks come from MediaPipe Hands, version 0.10.14 [20], and the classifier is the Random Forest [11] from scikit-learn [19]. OpenCV reads the frames and draws the output window, and NumPy does the array work for normalization. The laptop runs Windows. For the camera we streamed video from a phone over Wi-Fi with the IP Webcam app and read it as if it were a webcam. Table 3 lists the tools.

Table 3: Tools and their purposes

Tool (vendor)	Purpose
Python 3.11 (Python Software Foundation)	Programming language for the recognition pipeline
MediaPipe 0.10.14 (Google)	Hand landmark extraction
OpenCV	Reading video frames and showing the output window
NumPy	Storing and normalizing landmark data
scikit-learn	Training and evaluating the Random Forest
joblib	Saving and loading the trained model
IP Webcam (Android app)	Streaming the phone camera to the laptop
Visual Studio Code (Microsoft)	Writing and debugging the code

B. Dataset

We recorded the data ourselves. Public ASL alphabet collections are mostly images, whereas we needed landmark vectors captured with the same camera setup used at run time, so recording our own was simpler. While recording, we moved the hand a little (closer, farther, tilted, left, and right) to add variety. Each frame became one row: the letter label followed by 63 normalized values. We collected roughly 400 samples for each of the 24 static letters, which is about 9,600 samples in total. J and Z are left out because they are drawn in the air and cannot be told apart from a single frame. Because the wrist is the origin, its three values are always zero, so 60 of the 63 columns carry information. (See the below Figure 2)



	A	B	C	D	E	F	G	H	I	J	K	L
408	A	0	0	0	0.181474	-0.24426	-0.01165	0.252217	-0.54595	-0.00474	0.273015	-0.79318
409	A	0	0	0	0.17287	-0.2376	-0.01633	0.248218	-0.5387	-0.01532	0.275031	-0.78496
410	A	0	0	0	0.171984	-0.23267	-0.02047	0.247312	-0.5331	-0.021	0.272436	-0.77717
411	A	0	0	0	0.17983	-0.23292	-0.01675	0.256816	-0.53931	-0.0149	0.281762	-0.78604
412	A	0	0	0	0.192439	-0.22746	-0.00567	0.290059	-0.52182	0.002325	0.325153	-0.77184
413	A	0	0	0	0.200311	-0.22793	-0.01786	0.292854	-0.50735	-0.01398	0.321558	-0.75565
414	A	0	0	0	0.192321	-0.2086	-0.02035	0.285798	-0.49852	-0.01911	0.317341	-0.74034
415	A	0	0	0	0.189205	-0.19988	-0.0275	0.29035	-0.48657	-0.02912	0.321992	-0.73286
416	A	0	0	0	0.191261	-0.20216	-0.02044	0.290313	-0.48666	-0.01726	0.322586	-0.73391
417	A	0	0	0	0.19779	-0.20667	-0.01513	0.296907	-0.49837	-0.00973	0.328854	-0.73924
418	A	0	0	0	0.197498	-0.20315	-0.02195	0.298059	-0.49465	-0.01867	0.329569	-0.7393
419	A	0	0	0	0.19639	-0.18915	-0.02749	0.298214	-0.47694	-0.02694	0.330173	-0.73311
420	B	0	0	0	0.078884	-0.11501	-0.04143	0.117544	-0.29449	-0.05122	0.047751	-0.4109
421	B	0	0	0	0.077871	-0.12343	-0.04247	0.116071	-0.2948	-0.05362	0.045919	-0.41234

Figure 2: A part of the recorded dataset. Each row holds the letter label followed by the 63 normalized landmark values

C. Model Training

We split the data 80% for training and 20% for testing, and we did not touch the test part until the end. A Random Forest with N trees was trained on the 63-value vectors and saved with joblib. A Random Forest grows many decision trees on random subsets of the samples and features and takes a vote [11]. We picked it for four reasons: it works well on short numeric vectors, it needs little tuning, it trains quickly on a CPU, and it gives class probabilities that we use as the confidence value at run time.

D. Live Recognition

At run time every frame goes through MediaPipe, normalization, and the classifier. A letter is typed only when the same letter has been predicted for k frames in a row with a probability above T. Space and backspace are keyboard keys. If MediaPipe finds no hand, the system makes no prediction and resets the counter, so the text stays as it is.

V. RESULTS AND DISCUSSION

We report accuracy on the held-out test split and then look at why that number should not be read at face value.

A. Test accuracy

Accuracy is the share of test samples whose predicted letter equals the true letter. With $N_{correct}$ correct predictions out of N_{test} test samples, it is

$$\text{Accuracy} = N_{correct} / N_{test} \quad (2)$$

The classifier reached 70% on the held-out 20%. For 24 roughly balanced classes, guessing would give about 4%, so the model has clearly learned something, but 70% is not high enough to type with comfortably. We went through the wrong predictions to see which letters were mixed up most often.

B. Reading the 70% figure

The accuracy number needs care. Frames from one recording are nearly identical to their neighbors, so a random test split contains near-copies of training frames, and the test score probably flatters the model. In live use the

predictions were less steady than 70% suggests, and they changed with camera position, lighting, distance, and background. Koller [4] discusses how much results depend on the evaluation protocol, including whether the test signers also appear in the training data. A fairer test for our system would hold out whole recording sessions, or whole signers.

C. Confusable Letters

Letters made with a closed fist, such as M, N, S, and T, are usually the hardest to separate. They differ mainly in where the thumb sits against the fingers, and the thumb is often half hidden from the camera. Figure 3 shows the reference signs, and Figure 4 shows the prototype in use.

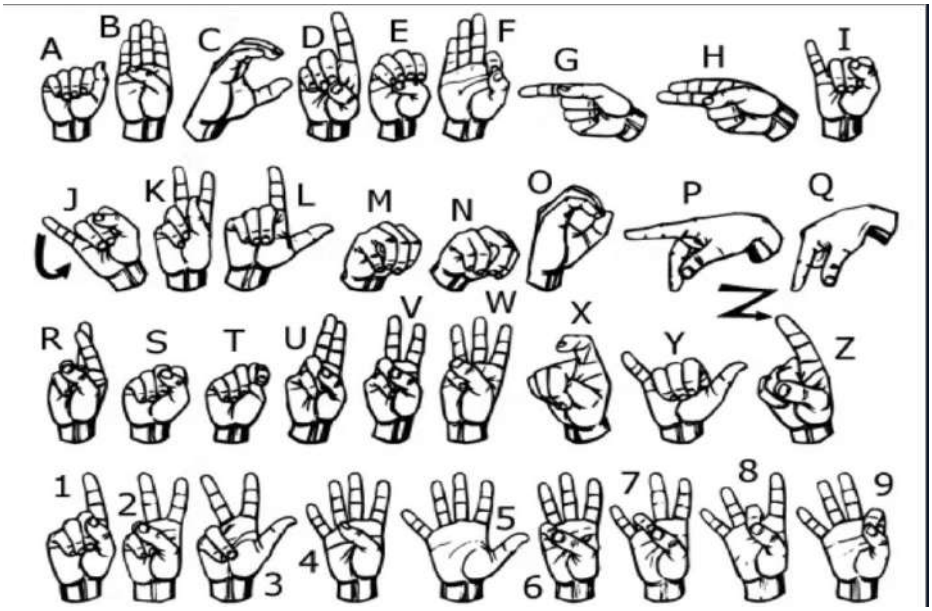


Figure 3: ASL hand signs for letters and digits. J and Z involve motion and are not recognized by the system



Figure 4: Prototype screenshots: (a) the interface, (b) and (c) detected signs with hand landmarks drawn on the frame.

D. Limitations

The system has clear limits. It covers 24 static letters only. All of the data were recorded by our team with one camera setup, so hand sizes, skin tones, lighting, and backgrounds are barely varied. There is no "no sign" class, so a hand that is not signing anything is still forced into one of the 24 letters. It tracks one hand and uses the one-handed ASL alphabet, so alphabets that use two hands are not covered. Finally, the 70% test score is probably optimistic, for the reasons given above.

E. Possible use and responsible use

Within these limits the system could help someone practicing fingerspelling, or serve as one part of a larger sign-to-text tool. It should not replace a human interpreter. Deaf signers have their own language and community, and we think they should be involved in designing, labeling, and testing tools like this one. We plan to ask for their feedback before claiming more.

VI. CONCLUSION AND FUTURE WORK

We set out to see how far a few free tools would get us on the smallest useful piece of sign language recognition, the fingerspelled alphabet. The answer is: far enough for a demonstration, not yet far enough to rely on. MediaPipe landmarks turned out to be a convenient input, and the Random Forest was easy to train and cheap to run. The weak point is generalization. A model trained on one camera setup and tested on near-duplicate frames says little about a new user in a new room, and the gap we saw between the test score and live use makes that plain.

Next we plan to add J and Z by classifying short sequences of frames, for example with an LSTM [15]; to record more people, cameras, lighting conditions, and backgrounds, and to test on signers the model has never seen; to add a "no sign" class; to report a per-letter confusion matrix and compare the Random Forest with other classifiers; to build a web or Android front end that can also translate the typed

text; and to move from letters toward words and sentences [14]. We would also like to try Indian Sign Language, which uses two hands for many letters. Throughout, we want deaf signers to check the labels and the usability.

DECLARATIONS

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Author Contributions

All authors contributed to the study conception and design. All authors read and approved the final manuscript.

Data Availability

The data supporting the findings of this study are available from the corresponding author upon reasonable request.

Funding

The authors received no financial support for the research, authorship, and/or publication of this article.

Declaration of Generative AI And AI-Assisted Technologies

The authors declare that generative AI and AI-assisted technologies were used only for language improvement, proofreading, and editing, where applicable. The authors take full responsibility for the accuracy, originality, and integrity of the content of this manuscript.

Ethics Approval

The study was conducted in accordance with applicable ethical guidelines

Acknowledgments

The authors would like to thank all individuals and institutions that provided support for this research.

REFERENCES

[1] World Health Organization, "Deafness and hearing loss," WHO Fact Sheet. Available from: https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss

[2] W. C. Stokoe, "Sign language structure: An outline of the visual communication systems of the American deaf," *Studies in Linguistics: Occasional Papers*, no. 8, Univ. of Buffalo, Buffalo, NY, USA, 1960. Available from: https://doi.org/10.1093/deafed/eni001

[3] H. Cooper, B. Holt, and R. Bowden, "Sign language recognition," in *Visual Analysis of Humans: Looking at People*, T. B. Moeslund et al., Eds. London, U.K.: Springer, 2011, pp. 539–562. Available from: https://doi.org/10.1007/978-0-85729-997-0_27https://doi.org/10.1007/978-0-85729-997-0_27

[4] O. Koller, "Quantitative survey of the state of the art in sign language recognition," arXiv:2008.09918, 2020. Available from: https://arxiv.org/abs/2008.09918

[5] R. Rastgoo, K. Kiani, and S. Escalera, "Sign language recognition: A deep survey," *Expert Syst. Appl.*, vol. 164, Art. no. 113794, 2021. Available from: https://www.sciencedirect.com/science/article/abs/pii/S095741742030614X

[6] N. Adaloglou et al., "A comprehensive study on deep learning-based methods for sign language recognition," *IEEE Trans. Multimedia*, vol. 24, pp. 1750–1762, 2022. Available from: https://ieeexplore.ieee.org/abstract/document/9393618

[7] N. Pugeault and R. Bowden, "Spelling it out: Real-time ASL fingerspelling recognition," in *Proc. IEEE Int. Conf. Comput. Vis. Workshops (ICCV Workshops)*, 2011, pp. 1114–1119. Available from: https://ieeexplore.ieee.org/abstract/document/6130290

[8] T. Starner, J. Weaver, and A. Pentland, "Real-time American Sign Language recognition using desk and wearable computer-based video," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 12, pp. 1371–1375, Dec. 1998. Available from: https://ieeexplore.ieee.org/abstract/document/735811

[9] K. Bantupalli and Y. Xie, "American Sign Language recognition using deep learning and computer vision," in *Proc. IEEE Int. Conf. Big Data*, 2018, pp. 4896–4899. Available from: https://ieeexplore.ieee.org/abstract/document/8622141

[10] Zhang et al., "MediaPipe Hands: On-device real-time hand tracking," arXiv:2006.10214, 2020. Available from: https://arxiv.org/abs/2006.10214

[11] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001. Available from: https://doi.org/10.1023/A:1010933404324

[12] S. Mitra and T. Acharya, "Gesture recognition: A survey," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 37, no. 3, pp. 311–324, May 2007. Available from: https://ieeexplore.ieee.org/abstract/document/4154947

[13] L. Pigou, S. Dieleman, P.-J. Kindermans, and B. Schrauwen, "Sign language recognition using convolutional neural networks," in *Computer Vision – ECCV 2014 Workshops*, Lecture Notes in Computer Science, vol. 8925. Cham, Switzerland: Springer, 2015, pp. 572–578. Available from: https://doi.org/10.1007/978-3-319-16178-5_40

[14] N. C. Camgoz, S. Hadfield, O. Koller, H. Ney, and R. Bowden, "Neural sign language translation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 7784–7793. Available from: https://ieeexplore.ieee.org/abstract/document/8578910

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997. Available from: https://doi.org/10.1162/neco.1997.9.8.1735

[16] Kapitanov, K. Kvanchiani, A. Nagaev, R. Kraynov, and A. Makhliarchuk, "HaGRID – HAnd Gesture Recognition Image Dataset," arXiv:2206.08219, 2022. Available from: https://arxiv.org/abs/2206.08219

[17] Lugaresi et al., "MediaPipe: A framework for building perception pipelines," arXiv:1906.08172, 2019. Available from: https://arxiv.org/abs/1906.08172

[18] V. Bazarevsky, Y. Kartynnik, A. Vakunov, K. Raveendran, and M. Grundmann, "BlazeFace: Sub-millisecond neural face detection on mobile GPUs," arXiv:1907.05047, 2019. Available from: https://arxiv.org/abs/1907.05047

[19] Pedregosa et al., "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[20] Google, "MediaPipe solutions guide." Available from: https://ai.google.dev/edge/mediapipe/solutions/guide