ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25apr2374

Enhancing Model Accuracy for Keypoint-Based Sign Language Recognition using Optimized Neural Network Architectures

Kailash Kumar Bharaskar[1]; Dharmendra Gupta[2]; Vivek Kumar Gupta[3]; Rachit Pandya[4]; Rachit Jain[5]

[1,2,3] Department of Computer Science, Medi-Caps University, Indore,India
[4,5] Student, Department of Computer Science, Medi-Caps University, Indore,India

Publication Date: 2025/05/17

Abstract: Sign language is a vital means of communication for millions, yet technological barriers still limit accessibility. To address this, we analyzed the existing deep learning model and identified key areas for enhancement. We expanded the neural network to improve learning capacity, replaced ReLU with LeakyReLU to avoid inactive neurons, and added batch normalization to maintain gradient stability throughout training. To reduce overfitting while preserving performance, we fine-tuned the dropout layer. We also improved preprocessing to filter out background noise, enhancing the system's ability to accurately track gestures. The training process was accelerated using early stopping and model checkpointing in order to save the best-performing version possible without incurring unnecessary computation. The final leg was converting the model to run in TensorFlow Lite, so that it would be able to run efficiently on mobile and edge devices and hence making its real-world deployment possible. The results were demonstrative; greatly improved accuracy, enhanced stability, and decent real-time performance. With confusion matrices and ROC curves backing it, the improvement is measurable. But more importantly, this project is about inclusivity—what it means to bring people into technology more finely on behalf of the community.

How to Cite: Kailash Kumar Bharaskar; Dharmendra Gupta; Vivek Kumar Gupta; Rachit Pandya; Rachit Jain (2025) Enhancing Model Accuracy for Keypoint-Based Sign Language Recognition using Optimized Neural Network Architectures *International Journal of Innovative Science and Research Technology*, 10(4), 4049-4055.https://doi.org/10.38124/ijisrt/25apr2374

I. INTRODUCTION

The sign language allows the deaf and hard-of-hearing community to convey complex thoughts, feelings, and concepts. But the lack of being familiar with sign language leaves obstacles to communication in everyday life. Automated Sign Language Recognition SLR Automated (SLR) is a significant development that utilizes machine learning to convert visual gestures into text or speech, facilitating smoother communication. The adoption of deep learning-based approaches has further augmented SLR systems, drastically enhancing their precision, speed, and overall robustness. Deep learning models are very good at identifying complicated patterns. The traditional SLR task based on vision typically leverages CNNs and RNNs. But there's an alternative — keypoint-based models. Rather than analyzing entire images, they follow hand joint movements which makes them lightweight and quick. They don't require as much computing power and are better equipped to handle background noise, making them ideal for real-time usage. That said, they present challenges. Overfitting. Poor generalization. Our research addresses these issues directly, we observed that this direct deployment of architectural improvements would yield models that were not only faster—but also more accurate and reliable in the real world. This research mainly aims to improve the accuracy and robustness of a keypoint-based sign language recognition model.

➢ *The Following Research Questions Guide Our Study:*

- How can neural network optimization techniques improve the accuracy of keypoint-based SLR models?
- What impact do modifications such as batch normalization, LeakyReLU activations, and adjusted dropout rates have on the model's stability and generalization capabilities?

II. LITERATURE REVIEW

In the last 10 years, sign language recognition (SLR) has evolved significantly. Researchers progressed from traditional methods — in which features such as those based on hand shape, movement, and timing were manually extracted from the data — to deep learning techniques, in which patterns in the data were automatically learned. Early systems had their moments, but they struggled with the complexity and variety of sign language as it is used in the

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25apr2374

real world. Gestures change. Context matters. And hand-rolled features could only do so much. That's where deep learning came into the picture, providing more flexible and scalable solutions.

The advent of deep learning has revolutionized the field of sign language recognition (SLR) enabling models to recognize gestures with remarkable accuracy. The CNNs are particularly good at extracting visual details — but they have their costs. They require enormous datasets and lots of computing power, making real-time use more difficult, especially on mobile devices. That's where the keypoint-based models play a role. Rather than full images, they report only positions of hands and joints. It is quicker, lighter and more adaptable. These models are better at handling different lighting, backgrounds and occlusions — and since they are less resource-heavy, they're a practical fit for real-world use, including on mobile and edge devices.

In a bid to comprehend the temporal nature of gestures in a sign language, Recurrent Neural Networks (RNNs), mostly Long Short Term Memory (LSTM) networks, have become popular. They're good at modeling sequences — how one sign progresses into another. Such approaches are ideal for continuous signing recognition, i.e., when there is a context. But they have challenges, too. RNNs are slow to train and have difficulty dealing with long sequences because of the vanishing gradient problem. Without those long frames, they may miss some important details, or lose track of earlier gestures.

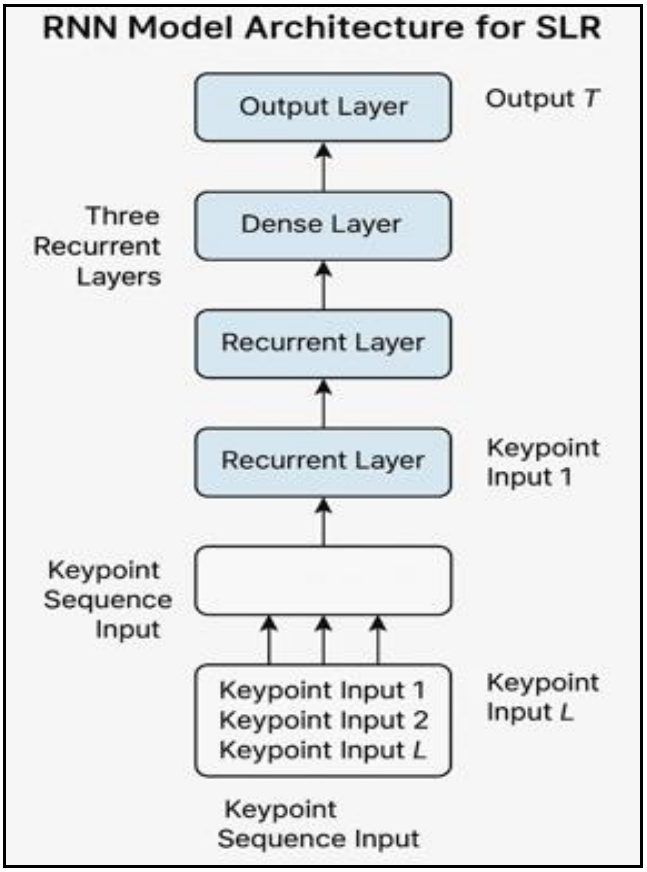

Fig 1 RNN Model Architecture for SLR

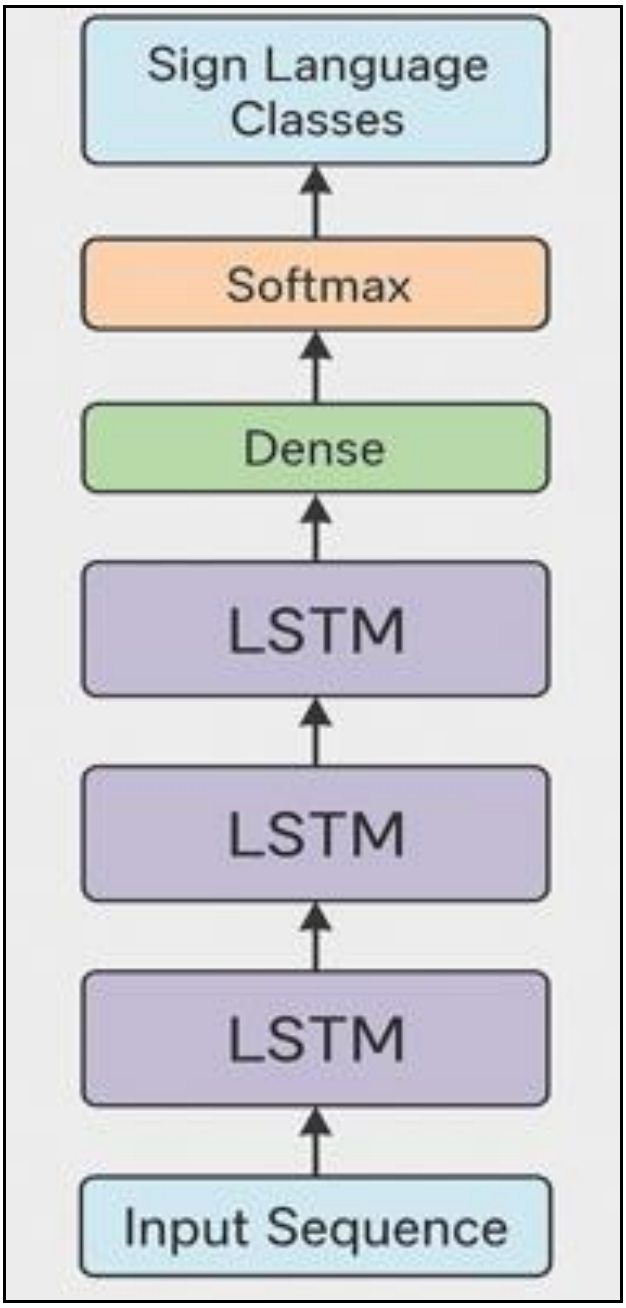

Fig 2 Model Architecture Diagram

Keypoint-based approaches take a more nuanced and targeted approach. Rather than analyzing a whole image, they track discrete points — such as joints in the hands and body — to determine what gesture is being made.Think of it like connecting the dots. Tools like OpenPose and MediaPipe make this easier by pulling out those keypoints accurately, even when the background is messy or the lighting isn't perfect. The big win? Less data to deal with. That means faster, lighter models that still perform well. And because they don't need a ton of computing power, they're a great fit for real-world use—especially on phones and small devices where speed and simplicity matter.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25apr2374

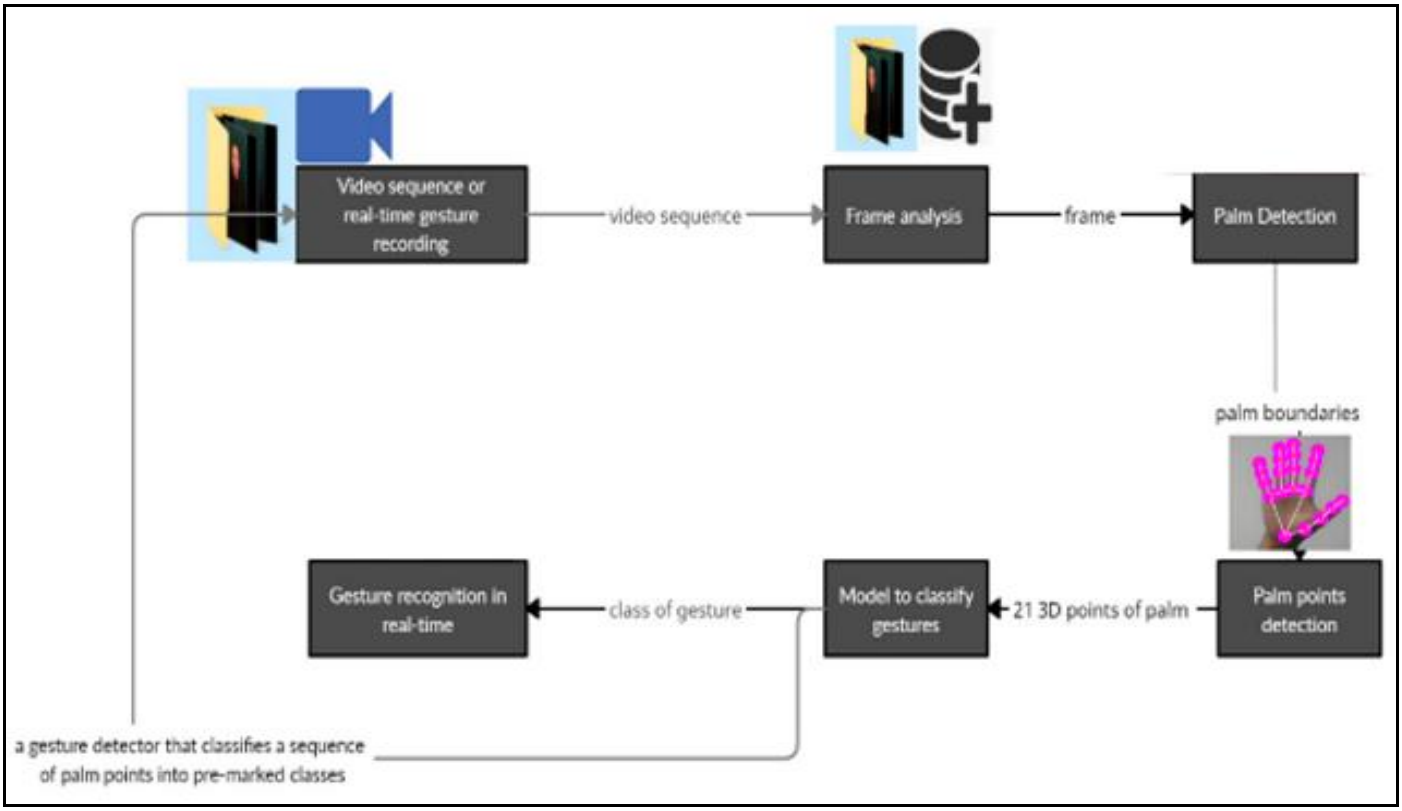

Fig 3 System Architecture for SLR

Sign language recognition (SLR) has really grown over the years. It started with simple machine learning models that used handcrafted features and basic classifiers like SVMs. They got the job done—sort of—but struggled with the complexity of real-world gestures. Then deep learning came along and changed the game. Early CNNs helped improve accuracy, but what really pushed things forward were techniques like dropout, batch normalization, and smarter activation functions. These made models not just better—but more stable, and actually usable outside the lab. In our work, we've built on all of that. We didn't just copy what came before—we dug into it. Looked at what worked. What didn't. Then focused on fine-tuning key parts of the model—like choosing the right activation functions, applying better normalization, and adjusting dropout levels. The result? A leaner, smarter network that learns faster and holds up better in the real world.

III. RELATED WORKS

➢ *Data Labelling and Structuring*

The first version of the sign language recognition system was built using TensorFlow and Keras, with a focus on keeping things simple and beginner-friendly. It worked by tracking 21 key points on the hand—so, 42 values for x and y coordinates—and running them through a few dense layers. ReLU was used for activation, and dropout helped reduce overfitting. The final softmax layer handled the actual gesture classification, sorting each input into a specific sign language category. Training was done using sparse categorical cross-entropy with the Adam optimizer. It did the job—but just barely. The model struggled with more complex patterns, overfitted easily, and wasn't exactly efficient when it came to training time. It was a good starting point, but clearly had room for improvement. In order to provide some performance improvements several important enhancements were made:

Increased Learning Power: Having more neurons in the dense layers could allow the model to learn more comprehensive features of the gesture. LeakyReLU Activation: Traditional ReLU activation was swapped to LeakyReLU, to mitigate the "dying neurons", encourage smoother gradient flow, and boost stability during training. In each layer image, we implement Input Normalization which allows the training process work faster and better(Batch Normalization). Optimized Dropout: After tuning dropouts several results were obtained which clearly measured the balance between overfitting and learning. Enhanced Data Processing: By adopting a more flexible data-loading approach, we could better handle inconsistencies, and feature standardization helped conform the features before learning. Lean Deployment: The model was converted to TensorFlow Lite leading to a decrease in size without sacrificing accuracy allowing the model to run on mobile and embedded device.

➢ *Training*

Optimizer & Learning Rate: Adam optimizer (LR = 0.001) based on the learning adaptable to data. Loss function: sparse categorical cross-entropy for multi-class classification. Regularization: Dropout (35%) to prevent overfitting. Instability in training backpropagation of batch normalization. Use Early Stopping to avoid computation time. Tracked. Metrics: Loss, accuracy and gradient flow through TensorBoard. Training Workflow: (Initialization: His initialization for stable weight distribution. Forward Pass:

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25apr2374

Processing keypoint data through the network. Loss Computation: Sparse categorical cross-entropy. Backward Pass & Weight Updates: Gradient updates via Adam optimizer. Validation: Performance evaluation of epoch).

IV. RESULTS

- The New Sign Language Recognition Model Improved Accuracy from 90.60% to 98.09% Proving the Classification Performance of Hand Gestures is Superior. Some Significant Observations from this Training Process Are:

- Training Accuracy Improvement: Throughout the training process, we observed that the model kept learning the distinguishing patterns from the target data, displaying its ability to identify the essential characteristics necessary to make accurate predictions.
- Good Generalization: The model was consistently performing well on the validation set, indicating that it was not overfitting the training set and it could also recognize new gestures which weren't part of the training dataset.
- Slow Decreasing of Loss: The training and validation losses go down slowly over time which indicates that we were learning well. This was further supported by techniques like dropout and batch normalization, which helped keep overfitting in check.
- Performance Comparison: Given in Fig 5

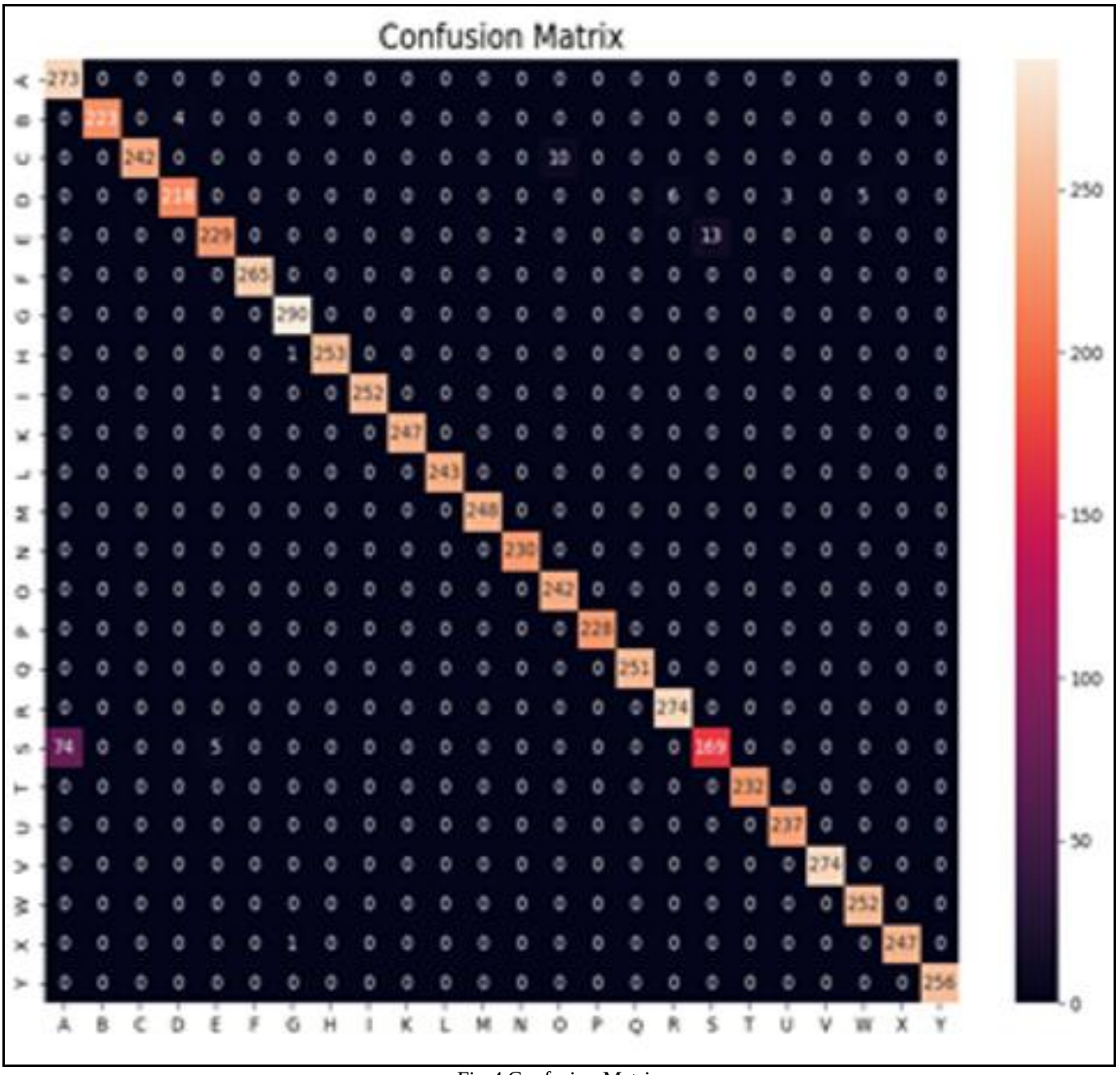

Fig 4 Confusion Matrix

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25apr2374

Component	Previous Model	Enhanced Model
Architecture Type	Sequential	Sequential
Input Shape	(42,) (21 keypoints × 2)	(42,) (21 keypoints × 2)
Initial Dropout	Dropout (rate = 0.2)	Dropout (rate = 0.2)
First Dense Layer	Dense (20 units, ReLU activation)	Dense (24 units, Linear activation) + LeakyReLU (α = 0.1)
Normalization Layer	None	BatchNormalization (applied after activation)
Second Dropout	Dropout (rate = 0.4)	Dropout (rate = 0.35)
Second Dense Layer	Dense (10 units, ReLU activation)	Dense (12 units, Linear activation) + LeakyReLU (α = 0.1)
Output Layer	Dense (NUM_CLASSES units, Softmax activation)	Dense (NUM_CLASSES units, Softmax activation)
Activation Strategy	ReLU activations directly inside Dense layers	Linear activations followed by LeakyReLU for gradient control
Regularization	Dropout only	Dropout + Batch Normalization
Parameter Count	Lower – simpler design with fewer neurons	Slightly higher – more units and normalization layers
Training Stability	Moderate – may need tuning for optimal generalization	Higher – more stable convergence due to normalization and flexible activations
Accuracy	90.60%	98.09%

Table 1 Performance Comparison

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25apr2374

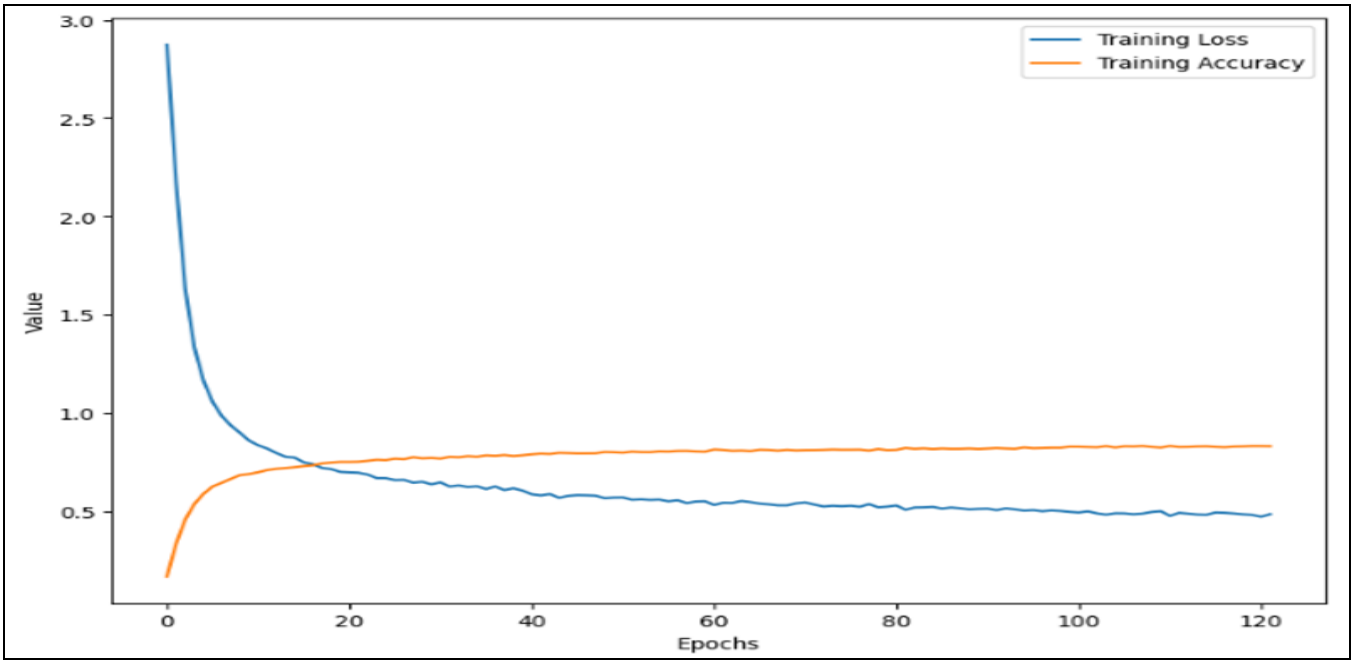

Fig 5 Epoch Diagram of Enhanced Model

V. LIMITATIONS

➢ *Although the Findings are Promising But Still We Have Number of Limitations Which are Given Below:*

- Limited Dataset Diversity: The training dataset may not represent the full spectrum of sign language variations found globally.
- Isolated Gesture Focus: The current system is designed for isolated gesture recognition, and additional work is needed to handle continuous signing.
- Real-World Variability: Factors such as occlusions, lighting variations, and background clutter in real-world environments can affect performance.
- Computational Trade-Offs: Despite optimizations, further work is needed to reduce model size and latency for deployment on ultra-low-power devices.

VI. CONCLUSION

In this work, we set out to build a smarter deep learning model for keypoint-based sign language recognition—and it paid off. By improving the network design, tightening up how we trained it, and cleaning the data that went in, we saw a clear jump in performance over the original version. We didn't just guess—we ran experiments, adjusted the settings, and visualized the results to see what really worked. Simple tweaks like adding batch normalization, switching to LeakyReLU, and fine-tuning dropout made a big difference in both accuracy and training stability.

But this isn't just about numbers on a chart. It's about people. By making it easier for machines to understand sign language, we're helping close the gap between deaf and hearing communities. It's a step toward better communication. And a more inclusive world.

FUTURE SCOPE

➢ *Future Research Should Address Several Critical Areas to Further Enhance Sign Language Recognition Systems:*

- Continuous Sign Language Recognition: Developing models that can handle continuous, fluid signing rather than isolated gestures.
- Multi-Modal Data Integration: Combining keypoint data with other sensory inputs (e.g., video, audio, depth) for richer feature representation.
- Real-Time Deployment: Further optimizing the model for real-time applications on mobile and embedded devices through advanced compression techniques.
- Robustness to Environmental Variability: Enhancing the system's resilience to changes in lighting, background, and occlusions using adaptive learning techniques.
- Ethical and Regulatory Considerations: Establishing frameworks for data privacy, user consent, and fairness to guide the development and deployment of SLR systems.

REFERENCES

[1]. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
[2]. Simonyan, K., & Zisserman, A. (2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv preprint.
[3]. Vaswani, A., et al. (2017). *Attention is All You Need*. Advances in Neural Information Processing Systems.\
[4]. Abeer et al., "Deep Learning for Sign Language Recognition: Current Techniques, Benchmarks, and Open Issues," ResearchGate, 2021.ResearchGate
[5]. SLR model-https://github.com/CodingSamrat/Sign-Language-Recognition

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25apr2374

APPENDIX

➢ *Workflow Diagram*

The below diagram shows how the enhanced model works (see fig 7.).

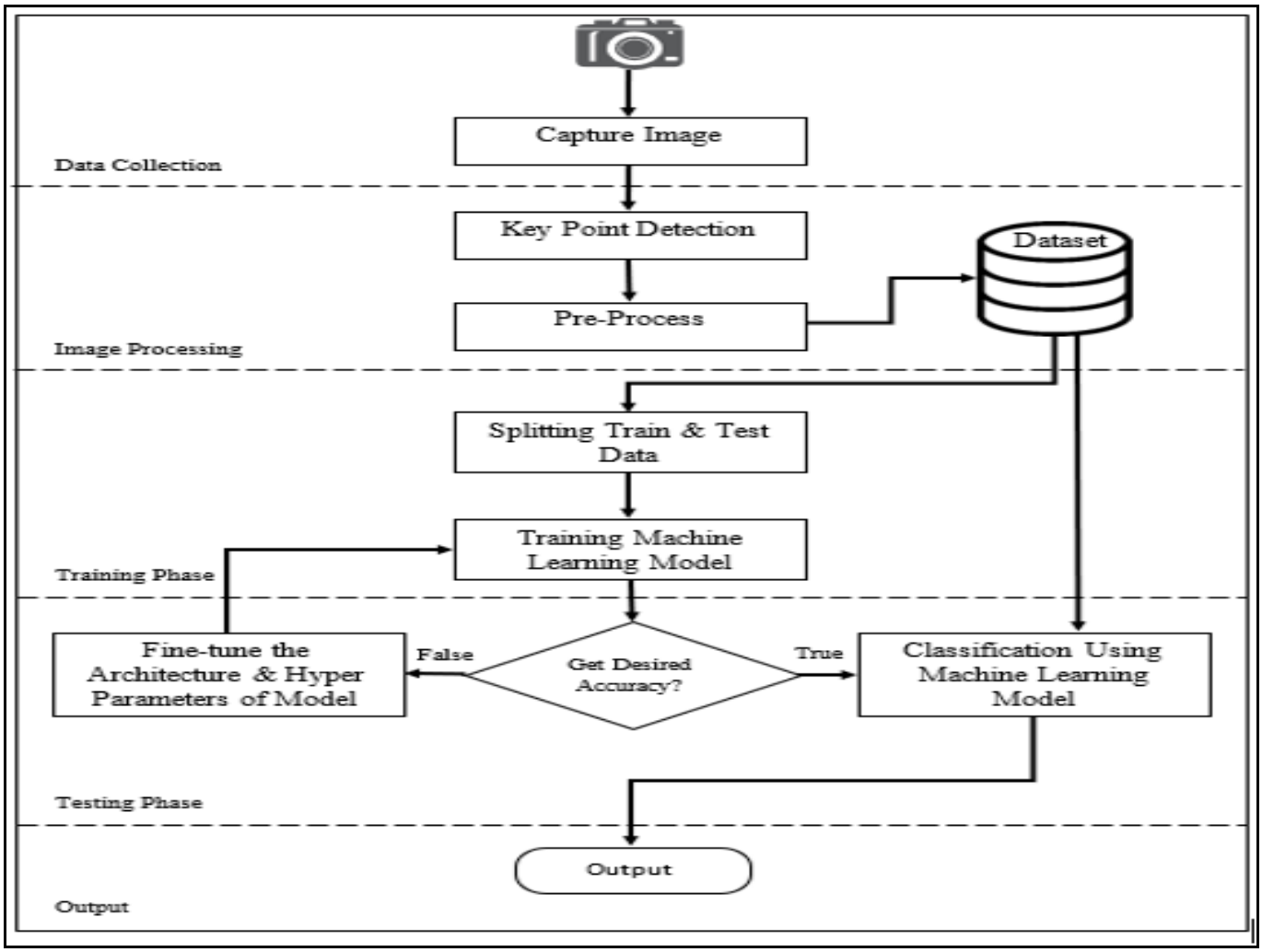

Fig 6 Workflow Diagram of Model

➢ *Data Preprocessing*

- Import numpy as np
- # using np.genfromtxt for robust data loading
- Dataset = '/path/to/keypoint.csv'
- X_dataset = np.genfromtxt (dataset, delimiter=',', dtype='float32', usecols=list (range (1, 43)))
- y_dataset = np.loadtxt(dataset, delimiter=',', dtype='int32', usecols=(0))
- # normalize keypoints
- X_dataset = (X_dataset - np.mean(X_dataset, axis=0)) / np.std(X_dataset, axis=0)

➢ *Model Architecture*

- import tensorflow as tf from tensorflow.keras.layers
- import Dense, Dropout, BatchNormalization, LeakyReLU, Input from tensorflow.keras.models import Sequential
- model = Sequential([
- Input(shape=(42,)),
- Dropout(0.2),
- Dense(24, activation='linear'),
- LeakyReLU(alpha=0.1),
- BatchNormalization(),
- Dropout(0.35),
- Dense(12, activation='linear'),
- LeakyReLU(alpha=0.1),
- Dense(NUM_CLASSES, activation='softmax')

➢ *Training Configuration*

- Model. compile (optimizer='Adam',
- loss='sparse_ categorical_ cross entropy',
- metrics=['accuracy'])
- History = model. Fit(Train, train,
- validation split=0.25,
- epochs=50,
- batch size=32,
- callbacks=[early stopping, model checkpoint])