Developing an Efficient and Lightweight Deep Learning Model for an American Sign Language Alphabet Recognition Applying Depth Wise Convolutions and Feature Refinement

Pillarisetty Uday Karthik[1*], Sai Subbarao Vurakaranam[2], Sumalatha M[3], Renugadevi R[4] and Sunkara Anitha[5]

{ udaykarthik58@gmail.com[1], saisubbarao373@gmail.com[2] , suma.magham@gmail.com[3], renu.rajaram@gmail.com[4], anithasunkara9@gmail.com[5]}

Department of CSE, Vignan's Foundation for Science Technology and Research, Guntur, India[1, 2, 3, 4, 5]

Abstract. In this, we proposed a deep learning framework for classifying American Sign Language (ASL) alphabet gestures to support accessibility for people with speech and hearing impairment. We evaluated our model using three public ASL datasets. one of the datasets of 87,000+ real-time images and 29 classes. To establish a baseline, we also tried state-of-the-art models, including VGG16, Efficient Net, MobileNetV1/V2, and ResNet50. Modeled after these findings, we created a compact application-specific convolutional neural network (CNN) model for static ASL recognition. Our custom ASL alphabet model employs depthwise separable convolutions, batch normalization, dropout, and global average pooling and adds a loop-based feature refinement block that is executed four times to increase spatial feature learning. Our test runs showed our model consistently achieved over 95% across various datasets while being faster and more reliable than the bulkier models. This model is a great candidate for real-time applications.

Keywords: sign language classification, deep learning, con- volution layers, CNN, Feature refinement.

1 Introduction

Only a fraction of society speaks sign language and those are the people born with a speech or hearing impairment. The lack of communication between the hearing majority and the deaf minority constitutes a serious barrier. Conversations are thus frequently slow, impersonal and inefficient. Here it can be imagined that in time of an emergency like a fireman needing to take a deaf person to safety a smartphone with a sign language recognition app installed can instantly be translated, allowing critical instructions to be transmitted in a timely manner. Similarly, the person could also send emergency information or requests to emergency personnel with the same tool.

This is human speak and for most people, it is the dominant mode of communication. The problem is that oftentimes non- or hard-of-hearing people have difficulty in speaking communication as well. Next is the sign language as it operates with hands, face and body to express a words or phrase rhythmically. However, there are also regional dialects and differences in sign languages which, like spoken languages, complicate the creation of an automatic translator system.

ICITSM-Part I 2025, April 28-29, Tiruchengode, India
Copyright © 2025 EAI
DOI 10.4108/eai.28-4-2025.2357809

In this paper, we leverage the power of Convolutional Neural Networks (CNNs) to automatically classify and interpret ASL alphabet hand signs. Inspired by prior art on hand detection and skin colour-based latent poses modelling, our work aims to increase the precision of sign language recognition using deep learning. The proposed system may eventually enable the provision and attainment of a better communication accessibility and quality of life for people with disabilities by teaching models' visual features of ASL signs.

Above the technical coverage, this research emphasizes the greater social significance of sign language recognition. Achievements in this domain also could be welcomed as part of inclusive communication systems for non-speaking or limited speaking persons. Although we concentrate our attention to ASL for this work, we recognize that the variation in the way gestures are executed by different users is a serious obstacle. Notwithstanding these challenges, the field has continued to move forward at a rapid pace, with deep learning methods showing potential in terms of ac- curacy and scalability. Through this work, we hope to help lay the groundwork for accessible technologies that support those who have speech and hearing impairments and enable them to participate more fully in society.

2 Related Work

Sabeenian et al. [10] utilized a dataset with 27,455 training and 7,172 validation images with 784 features for each image. They proposed a CNN-Based algorithm that uses an adapted CNN network with 11 layers, which contained the layers of the convolutional, max-pooling, flatten, dense, and dropout. It resulted in model with a training accuracy of 99% and a validation accuracy of 93% which is an excellent performance in a ASL gesture recognition.

Barbhuiya et al [2]. In their approach, they utilized the American Sign Language (ASL) dataset. The dataset is divided into 36 classes of sign characters which are the combination of 26 alphabetic and 10 numerical classes. The model architecture includes pre- processing to resize images to 227×227×3 and 224×224×3 for AlexNet and VGG16, respectively. Model: The VGG16 of layers contains convolutional layers of various filter sizes and Pooling layers, concluding with fully connected layers. The system obtained excellent accuracy in ASL sign recognition through pre-trained AlexNet and VGG16 models with SVM classification.

Sapriya et al [3]. They employed the IISL2020 dataset as part of their method. In their work, they proposed a model based on Keras and TensorFlow libraries that used a mixture of LSTM and GRU models and was trained on a Super Server workstation with dedicated hardware. They were able to achieve a high accuracy of 97% in the identification of Indian Sign Language.

Syamala et al [5]. For this, they have employed a dataset containing 200 Indian sign language words signed by 5 native ISL individuals. They applied the techniques of training CNN in a combination of 3 samples and testing in the rest of the 2 samples. The model applies the tanh activation function with bias add and stochastic pooling. The recognition rates for CNN architecture appear to perform better than other classifiers such as Ada boost, ANN.

The work by Aditya Das et al [4] including the dataset of RGB and depth images of320x240 resolution with hand gesture along 140 classes includes finger-spelling numbers and alphabets. The techniques adopted are CNN for feature extraction and ANN for classification. The model

used is the Inception V3 and its accuracy of 91.7% shows high penetration in sign language recognition.

3 Methodology

3.1 Sign Language Dataset

You need special ASL datasets to train a sign language recognition model. Additional datasets: the American Sign Language Dataset on Kaggle for image classification, DAI - ASL LVD dataset with videos with synchronised transcriptions and lexical annotations, WLASL dataset recording 2,000 frequent ASL words and How2Sign which contains more than 80h of sign language videos (also multi-modal). Fig 1 Sample Images of American sign language.

One of the ASL dataset we used has 29 classes out of 87,000 images (char from A to Z, Delete, space or Nothing). We used a total of 87000 images from GNIRD for training, out of which we separated 7830 for testing to help us fine-tune the final model.

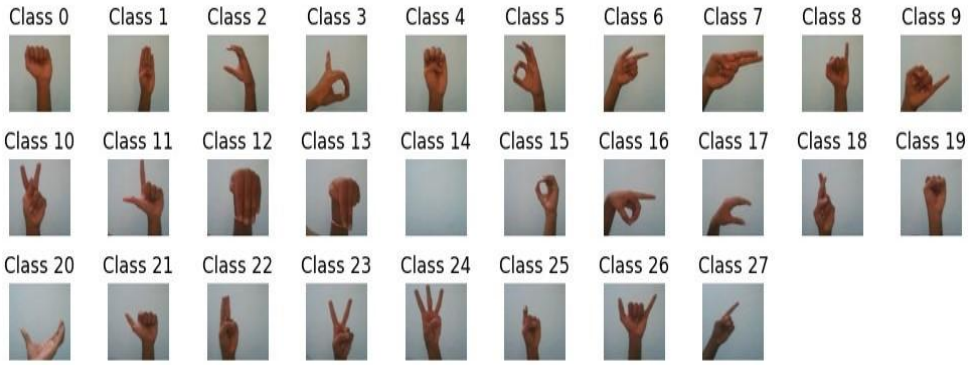

Fig. 1. Sample Images of American sign language.

3.2 Implementation Flow

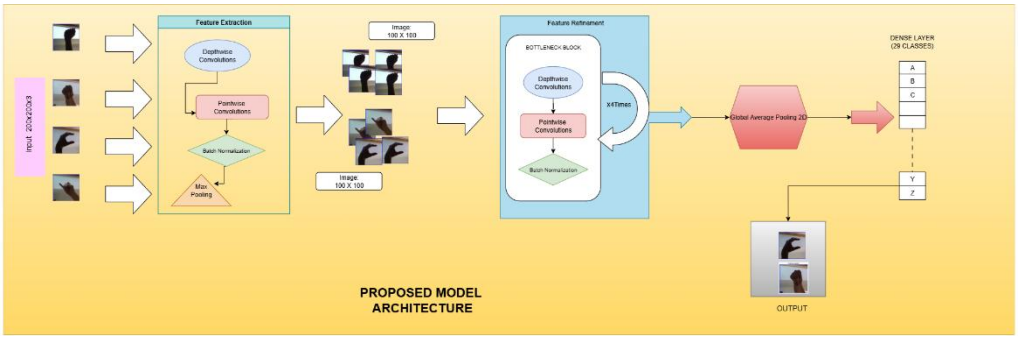

Fig. 2. Direction Flow of implementation.

Fig 2 shows this process, starting from the literature review, the definition of the research goal. This action is about reading related works in order to gain knowledge of methods and problems used in the research of sign language recognition. Then the problem finding procedure further serves to identify the major research goals. Then data collection is conducted to obtain sign

language gesture datasets. This information is then sent through a pre-processor where any pre-processing can be done, such as forcing behaviour relative to the model to allow for better model performance. Depth wise convolutions are employed for feature extraction to capture essential gesture patterns. The features extracted are then inputted to a Modified MobileNetV2 model which is also tuned for sign language recognition. Before it can be used to predict, the model must be trained and tested to ensure that it is appropriately accurate. The output is the signed language gestures which is ultimately helping in the communication process for speech and hearing-impaired people.

3.3 Data pre-processing

Data pre-processing is an important process in data pipeline analysis which is used to cleanse the input data so that it becomes better quality and can be passed as input to the data mining algorithm. It will organize and label the dataset so that it is more structured, thus creating a better opportunity for machine learning to learn from. For instance, your data real world data might be high signal-to-noise ratio and pre-processed in a way as to select relevant features or filter out noise or artefacts; this is fine for some kinds of datasets provided that it represents accurately the actual best version of that data possible but not OK if the distributional shift between training set and test set causes targeting bias. E.g., values will probably be normalized and the feature selection should be parts of the learning process rather than pre-processing steps.

3.4 Data Augmentation

In machine learning, data augmentation is the creation of new training samples using only the original dataset. Augmentation of data will augment the training set by creating modified versions of a given dataset. This manipulation can sometimes mean changing the original data, or be through creating what appear to be legitimate new data points via deep learning. Data Augmentation: transforms images to correct for overfitting by performing several transformations on the images. We adopted the rescaling here. Normalization of the pixel data is applied by rescaling. The pixel contains value between (0 – 255). With rescaling, my pixels would have been in the range 0-1.

We also used image resizing. The shape of image is 200,200,3. The size of the image has been reduced for sizing down the system compatibility, since the data is large, in initial flow using max pooling.

3.5 Mobile Net V2 Model Setup

The depth of the CNN model is 26. The SSD (Single Shot Multibox Detector) is a detection model with Mobile Net v2 as a backbone and it is efficient allowing to carry it out in real time. In this architecture, input image of size 300x300 is considered for processing. The backbone of Mobile Net v2 is depth wise separable convolutions that are used to produce multi-scale feature maps, which makes the model better capture features at different scales. Moreover, multiple feature maps at various scales are able to detect objects of various sizes. A detection network in the SSD combined with differently shaped and scaled anchor boxes to increase the precision of detection. The model, for each anchor, attempts to predict both the class of object and refine the positions of the bounding boxes, so that objects can be accurately placed in the image.

$$Classification\ Loss(L_{cls}) = \sum_{i(x,y,w,h)} smooth(l_i - g_i) \quad (1)$$

where c1 is the ground truth class and P(xi) is the predicted probability.

$$Localization\ Loss(L_{loc}) = \sum_{i(x,y,w,h)} smooth_{L1}(l_i - g_i) \quad (2)$$

$$Total\ Loss(L_{total}) = L_{cls} + L_{loc} \quad (3)$$

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 200, 200, 32)	896
batch_normalization (BatchNormalization)	(None, 200, 200, 32)	128
depthwise_conv2d (DepthwiseConv2D)	(None, 200, 200, 32)	320
batch_normalization_1 (BatchNormalization)	(None, 200, 200, 32)	128
conv2d_1 (Conv2D)	(None, 200, 200, 64)	2112
batch_normalization_2 (BatchNormalization)	(None, 200, 200, 64)	256
max_pooling2d (MaxPooling2D)	(None, 100, 100, 64)	0
depthwise_conv2d_1 (DepthwiseConv2D)	(None, 100, 100, 64)	640
batch_normalization_3 (BatchNormalization)	(None, 100, 100, 64)	256
conv2d_2 (Conv2D)	(None, 100, 100, 128)	8320
batch_normalization_4 (BatchNormalization)	(None, 100, 100, 128)	512
max_pooling2d_1 (MaxPooling2D)	(None, 50, 50, 128)	0
depthwise_conv2d_2 (DepthwiseConv2D)	(None, 50, 50, 128)	1280
batch_normalization_5 (BatchNormalization)	(None, 50, 50, 128)	512
conv2d_3 (Conv2D)	(None, 50, 50, 128)	16512
batch_normalization_6 (BatchNormalization)	(None, 50, 50, 128)	512
depthwise_conv2d_3 (DepthwiseConv2D)	(None, 50, 50, 128)	1280
batch_normalization_7 (BatchNormalization)	(None, 50, 50, 128)	512
conv2d_4 (Conv2D)	(None, 50, 50, 128)	16512
batch_normalization_8 (BatchNormalization)	(None, 50, 50, 128)	512
depthwise_conv2d_4 (DepthwiseConv2D)	(None, 50, 50, 128)	1280
batch_normalization_9 (BatchNormalization)	(None, 50, 50, 128)	512
conv2d_5 (Conv2D)	(None, 50, 50, 128)	16512
batch_normalization_10 (BatchNormalization)	(None, 50, 50, 128)	512
depthwise_conv2d_5 (DepthwiseConv2D)	(None, 50, 50, 128)	1280
batch_normalization_11 (BatchNormalization)	(None, 50, 50, 128)	512
conv2d_6 (Conv2D)	(None, 50, 50, 128)	16512
batch_normalization_12 (BatchNormalization)	(None, 50, 50, 128)	512
depthwise_conv2d_6 (DepthwiseConv2D)	(None, 50, 50, 128)	1280
batch_normalization_13 (BatchNormalization)	(None, 50, 50, 128)	512
conv2d_7 (Conv2D)	(None, 50, 50, 256)	33024
batch_normalization_14 (BatchNormalization)	(None, 50, 50, 256)	1024
global_average_pooling2d (GlobalAveragePooling2D)	(None, 256)	0
dense (Dense)	(None, 29)	7453

Fig. 3. MobileNet V2 architecture.

Fig 3 shows the MobileNetV2 architecture. The Table 1 represents the model summary of the proposed modified MobileNetV2 based CNN model. The model consists of a series of convolutional layers, depth wise separable convolutions where it includes pointwise convolutions, batch normalizations, and pooling layers. It retains the efficiency of MobileNetV2

while optimizing performance for classification tasks. The total number of parameters is 132,125, with 118,669 trainable parameters.

3.6 Proposed lightweight CNN Model

Table 1. Architecture of The Proposed Custom CNN Model.

Layer Type	Details	Repetitions
Input Layer	Input shape: (200, 200, 3)	1
Conv2D	3×3 filters, ReLU activation	5
DepthwiseConv2D	3×3 filters, ReLU activation	4
Batch Normalization	After conv/depth wise layers	9
Max Pooling2D	Pool size: 2×2	2
Dropout	0.3–0.5 rate	4
Global Average Pooling2D	Reduces spatial dimensions	1
Dense (Output)	29 units (SoftMax)	1

For compiling, we used activation functions ReLU and SoftMax in the model and we also used Early Stopping. The optimizer we used in the model is Adam. After performing some iterations, we achieved 99.7% of accuracy.

3.7 Batch normalization

Batch normalization leads to better training consistency, faster training and better accuracy in your model. It also protects against the problems with the gradients being too large or too small as you were training and helps avoid get stuck or making catastrophic mistakes learning. It is also beneficial for a neural network training that may struggle rated with long time if training/retraining, unstable gradients, and slow training.

3.8 Early Stopping

Early stopping is a regularization method to avoid overfitting the neural network by monitoring the model performance with a validation set. It makes it less susceptible to overfitting, and you are sure that the model will not learn any noise from the training data.

3.9 Activation Functions

Neural networks made activation functions as a crucial element because these carries non-linearity in output signal of respective neurons. Activation functions also work on the weighted sum of an input with bias term to produce an output signal it passes to the following layers, analogously. These patterns and relationships could be replicated using neural networks. ReLU, Soft max are the two activation functions we used in our model.

3.9.1 ReLU

The Rectified Linear Unit (ReLU) is among the most used activation functions in neural networks. ReLU adds non-linearity by returning the input directly if positive, and zero otherwise.

This simple and effective mechanism allows the model to learn complex representations and improves the efficiency of learning throughout deep architectures.

$$f(x) = \max(0, x) \tag{4}$$

3.9.2 SoftMax

The SoftMax activation function transforms raw outputs of a neural network (raw values, also known as logits), into a probability distribution with each individual element representing the likelihood of a class in comparison to the other classes.

$$Softmax(x_i) = \sum \frac{exp(x_i)}{jexp(x_j)} \tag{5}$$

3.10 Optimizer

They are designed as optimization algorithms of learnable parameters in a model like weights, biases to minimize loss and improve the model. The optimizer determines how the weights of a model are modified in the training process. In this work, we used Adam optimizer, a popular optimizer with good convergence and adaptive learning rate.

3.10.1 Adam Optimizer

Adam is a high order optimization technique which combines the benefits of Root Mean Square Propagation and Adaptive Gradient Algorithm. On the other hand, Adam has a separate learning rate for each parameter which reduces the oscillations and speeds up convergence. For example, in traditional gradient descent the learning rate is a constant, and it may converge slower (if at all) due to oscillation. The second optimizer is the learning optimizer that would be suitable for deep neural network by utilizing momentum and RMS techniques incorporated into one. Adam optimizer update rule:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{6}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{7}$$

$$\widehat{m}_t = \frac{m_t}{(1 - \beta_1^t)} \tag{8}$$

$$v\hat{}_t = \frac{v_t}{(1-\beta_2^t)} \tag{9}$$

$$\theta_{t+1} = \theta_t - \sqrt{\frac{\eta}{v\hat{}_t + \epsilon}} \widehat{m}_t \tag{10}$$

Where:

- θ_t is the parameter at time step t.
- g_t is the gradient of the objective function with respect to θ_t.
- β_1 and β_2 are decay rates for the moment estimates.
- m_t and m_t are first and second moment estimates, respectively.
- η is the learning rate.
- $\in$ is a small constant to prevent division by zero.

4 Results

4.1 Prediction

We have done our project to classify properly to predict the images that have been given to test the performance of the model. The results are shown below, where the model correctly predicted a sample input symbol as 'Y' with a probability of 100 %.

The predicted symbol and probability according to each class is below in Fig 4, 5 and 6.

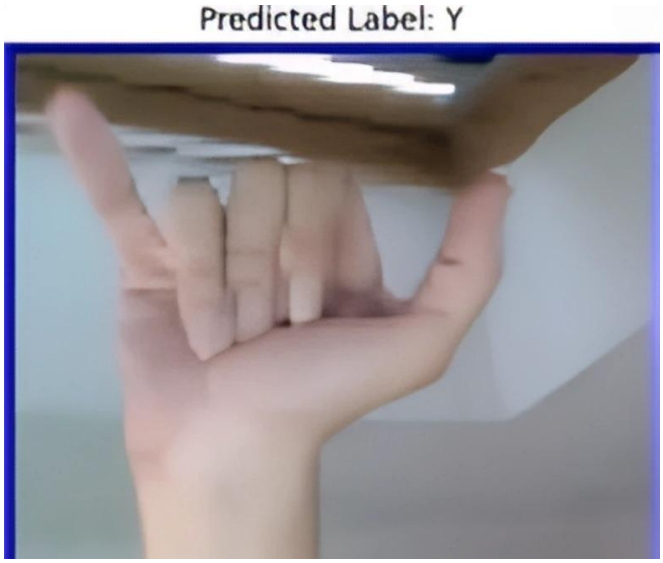

Fig. 4. Probability of predicted 'Y' labelled data.

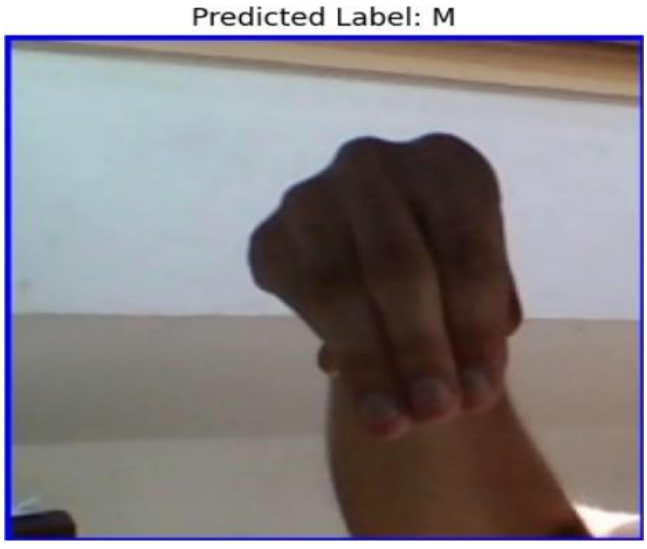

Fig. 5. Probability of predicted 'M' labelled data.

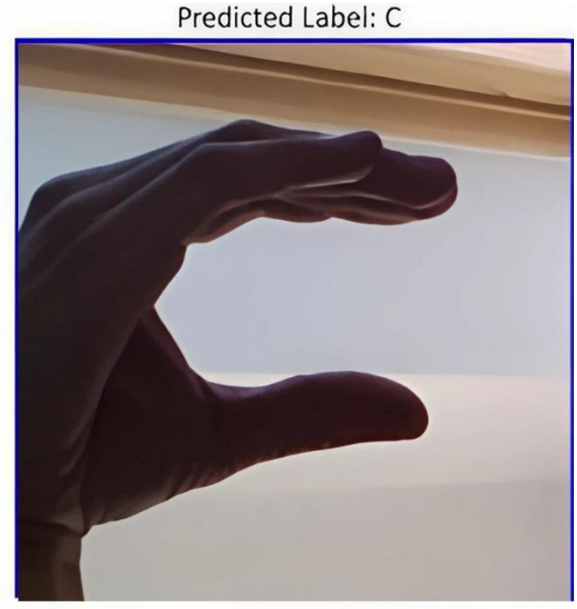

Fig. 6. Probability of predicted 'C' labelled data.

In our case, our model is a lightweight model, which is more efficient, and we use the Adam optimizer. The loss function was drastically reduced over 10 epochs, dropping below 0.09. The validation accuracy increased by 3% from the first epoch to 100% at the end of the last 10th epoch. Fig. 7 shows the Train and validation accuracy curves for asl-alphabet dataset.

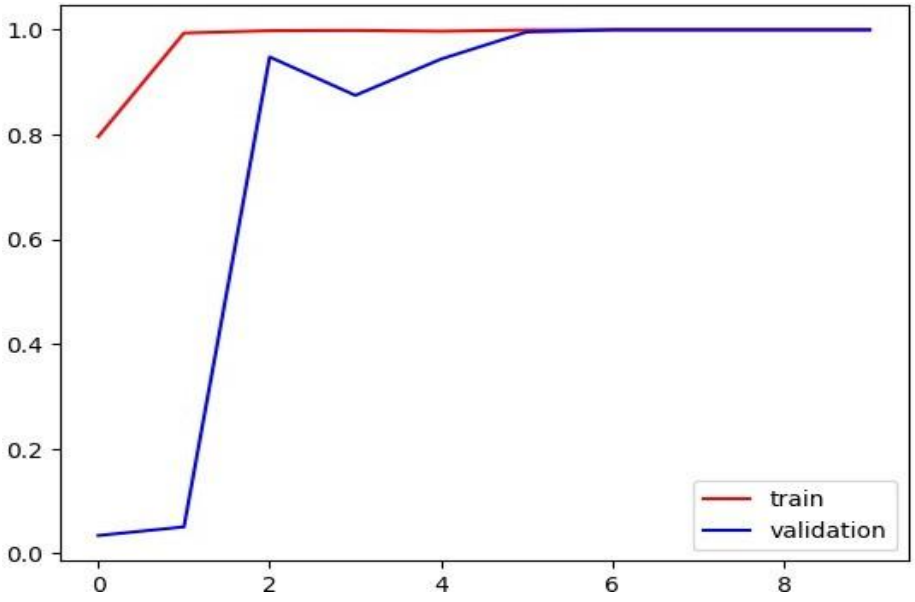

Fig. 7. Train and validation accuracy curves for asl-alphabet dataset.

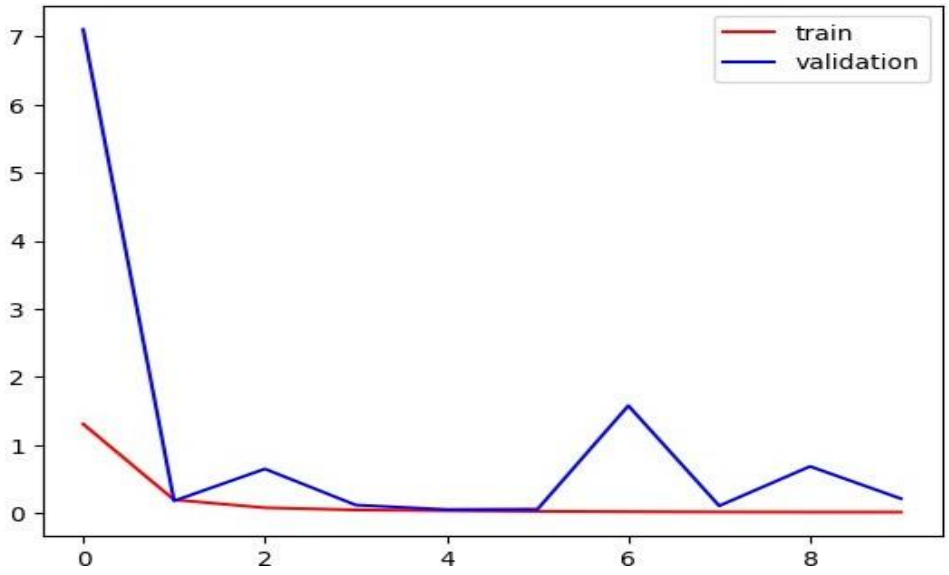

Fig. 8. Train and validation loss curves for asl-alphabet dataset.

As the epochs increase, we are able to obtain the average of 99 % validation accuracy, which is the greatest of all of those for our model. Fig 8 shows the Train and validation loss curves for asl-alphabet dataset.

Table 2. Different Model Comparison.

Model Name	Parameters	Epochs	Learning Rate	Training Accuracy	Validation Accuracy
VGG16	107,116,381	15	0.001	48	34.56
ResNet50	23,647,133	15	0.001	99.87	99.73
EfficientNet	4,086,713	15	0.001	99.94	98.02
MobileNetV2	2,295,133	15	0.001	99.07	94.13
Proposed Model (Custom MobileNetV2)	132,125	15	0.001	100	99.94

Table 2 shows the Comparison of Different Model Performance. A number of deep learning models, including VGG16, ResNet50, EfficientNet B0, and MobileNetV2, as well as suggested MobileNetV2 variants, are compared. Over 15 epochs, it highlights important parameters like total parameters, learning rate, training accuracy, and validation accuracy. With the highest validation accuracy of 99.94%, the suggested custom MobileNetV2 demonstrates the effectiveness of its model optimization.

Table 3. Model with dropouts for different learning rate Comparison.

Model Name	Parameters	Epochs	Learning Rate	Training Accuracy	Validation Accuracy
Proposed Model with dropouts	132,125	15	0.002	99.53	92.14
Proposed Model with dropouts	132,125	15	0.0001	99.82	94.97

Table 3 examines what happens when we increase learning rate with added dropout layers to the model architecture recommended earlier. Both set-ups have 132,125 parameters and were trained for 15 epochs. The model trained at 0.0001 learning rate acquires a better validation accuracy of 94.97% as opposed to the 92.14% with another model at the 0.002 learning rate, illustrating the importance of hyperparameter tuning in improving generalization of models accomplished by SOTA techniques.

Table 4. Performance Comparison of Different Models on Asl Alphabet Dataset.

Model	Accuracy	Precision	Recall	F1 Score	Accuracy
Proposed Model	0.99	0.93	0.87	0.87	0.99
MobileNet V1	0.71	0.91	0.88	0.88	0.71
MobileNet V2	0.95	0.91	0.88	0.88	0.95
VGG16	0.48	0.95	0.93	0.93	0.48
ResNet50	0.92	0.96	0.94	0.94	0.92
EfficientNet	0.89	0.92	0.89	0.89	0.89

We evaluated the Enhanced MobileNet model on the American Sign Language dataset and ASL dataset along with our primary datasets. Table 4 shows the performance comparison of different models on ASL Alphabet Dataset. Fig 9 shows the train and validation accuracy curves for the ASL dataset.

One of each the 28 classes is mapped to via a given color image in the American Sign Language Dataset thereby correspondingly representing each letter of the ASL alphabet Because these photos were taken in different light conditions and against different backgrounds, this database can be used to examine how well the model generalizes. Train and validation loss curves for the ASL dataset as shown in the Fig 10. American sign language · Fig 11 shows training and validation accuracy curves for models on the American sign language dataset. You will get 2,515 verified images comprised of 36 different classes ASL Dataset This dataset although of less instances are having a wider class distribution and can be tested on low resource multi-class classification.

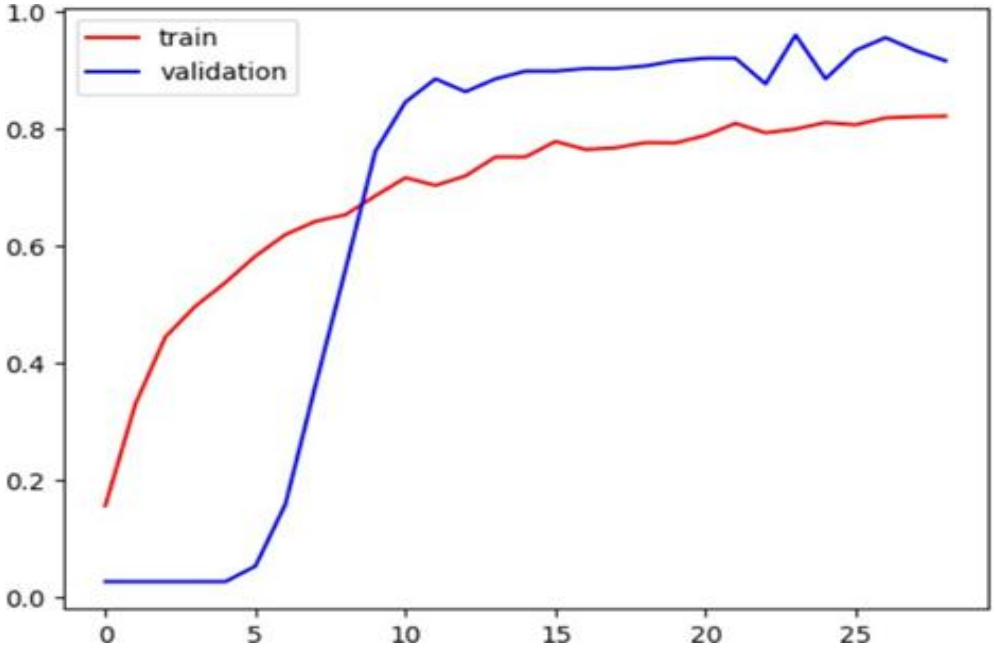

Fig. 9. Train and validation accuracy curves for asl-dataset.

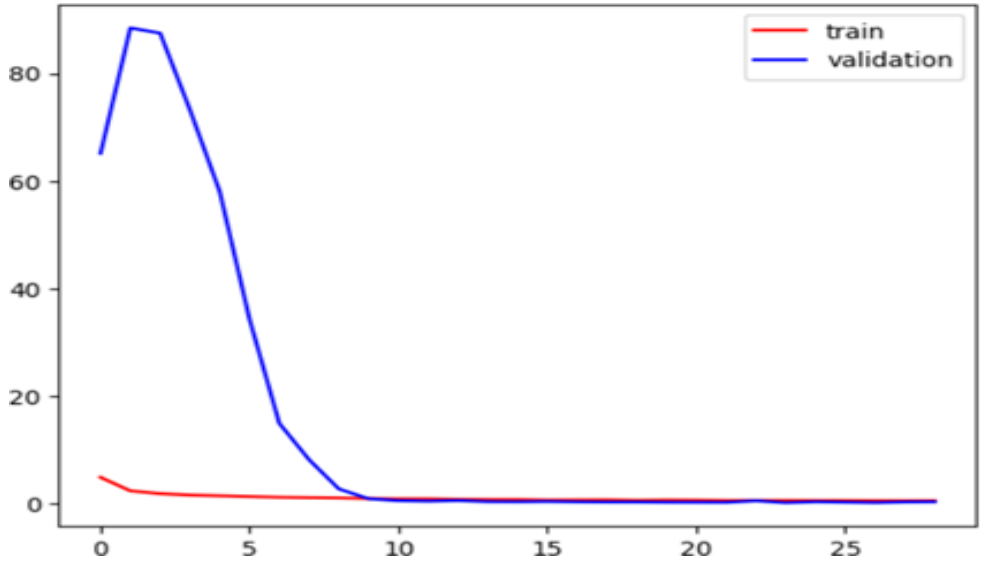

Fig. 10. Train and validation loss curves for asl-dataset.

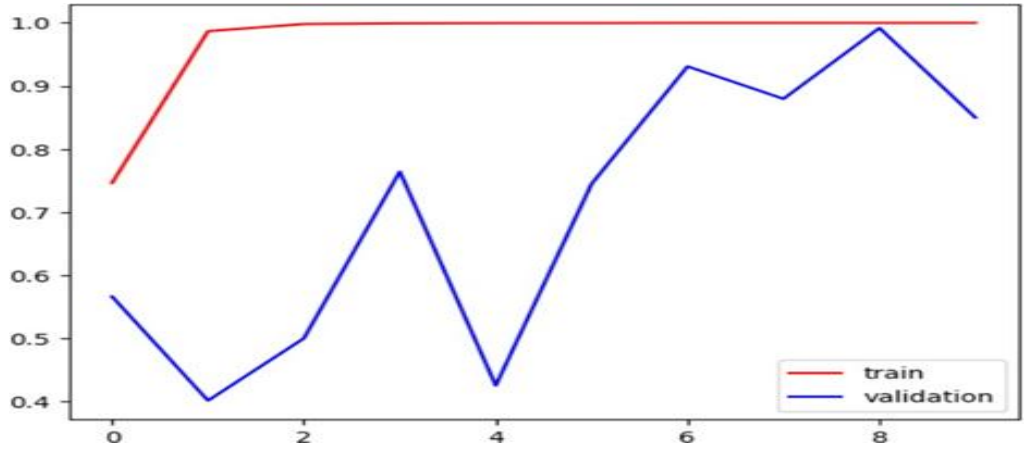

Fig. 11. Train and validation accuracy curves for American sign language dataset.

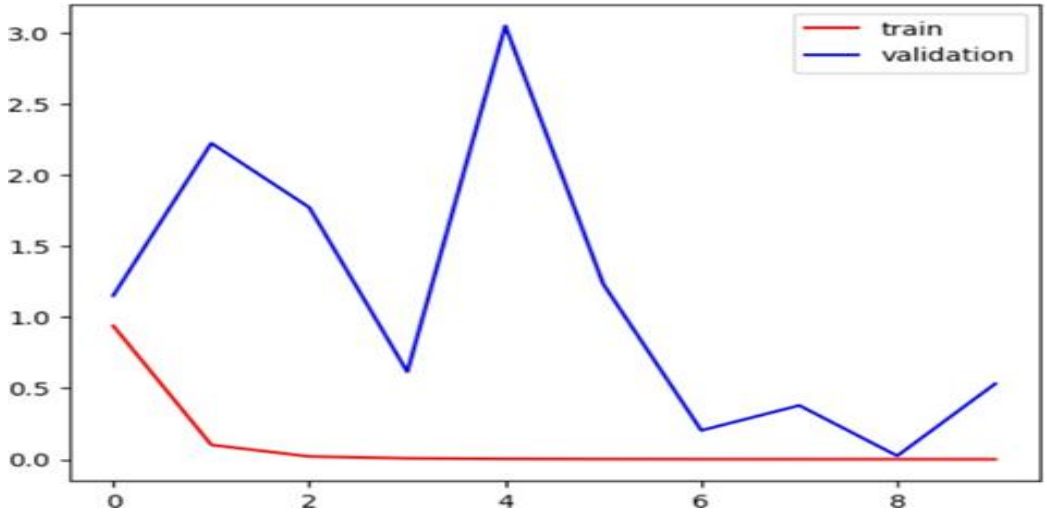

Fig. 12. Train and validation loss curves for American sign language dataset.

Table 5. Performance comparison of Enhanced MobileNet model on different ASL Alphabet dataset.

Model	Accuracy	Precision	Recall	F1 Score	Accuracy
asl-alphabet	0.99	0.93	0.87	0.87	0.99
asl-dataset	0.96	0.92	0.85	0.86	0.96
American-sign-language	0.85	0.92	0.85	0.86	0.85

Table 5 depicts the Comparison of performances of the proposed Enhanced MobileNet model on three different ASL datasets. The accuracy of the model reaches 0.99 on the ASL Alphabet dataset, where the standardized and well-defined sign images shows an excellent performance. The ASL Dataset and American Sign Language Dataset exhibit slightly less accuracies of 0.96 and 0.85 as the cause of greater variation on the signer's posture, background conditions, and the dataset variety. In spite of these variations, the model achieves high precision on both datasets, which demonstrates the robustness of the model in the classification of hand gestures whenever a prediction is performed. Fig 12 shows the Train and validation loss curves for American sign language dataset.

5 Conclusion and Future Scope

In order to effectively classify the fixed hand symbols of the American Sign Language (ASL) letters, we created a custom Convolution Neural Network (CNN) model based on MobileNet. The effectiveness of the pre-processing procedures and the used dataset is demonstrated by the developed model's excellent testing accuracy, which exceeded 99% on image data. The outcomes validate the CNN's resilience and efficiency in classifying ASL gestures, as well as its potential for incorporation into actual assistive communication systems.

6 Acknowledgement

We want to express our gratitude to open-source resources such as the Keras and TensorFlow communities, as well as internet resources like OpenAI and other artificial intelligence, for their invaluable contributions to our learning process.

References

[1] Singh, G., Yadav, A. L., S e h g a l , S. S. (2022). Sign language recognition Using Python. Interna- tional Conference on Cyber Resilience, ICCR 2022. https://doi.org/10.1109/ICCR56254.2022.9996001

[2] Barbhuiya, A. A., Karsh, R. K., Jain, R. (2021). CNN based feature extraction and classification for sign language. Multimedia Tools and Applications, 80(2), 3051–3069. https://doi.org/10.1007/s11042-020-09829-y

[3] Kothadiya, D., Bhatt, C., Sapariya, K., Patel, K., Gil- Gonza´lez, A. B., Corchado, J. M. (2022). Deep- sign: Sign Language Detection and Recognition Us- ing Deep Learning. Electronics (Switzerland), 11(11). https://doi.org/10.3390/electronics11111780

[4] Das, A., Gawde, S., Suratwala, K., Kalbande, D. (n.d.). Sign Language Recognition Using Deep Learning on Custom Processed Static Gesture Images.

[5] Rao, G. A., Syamala, K., Kishore, P. V. v, Sastry, A. S. C. S. (n.d.). Deep Convolutional Neural Networks for Sign Language Recognition.

[6] Nirmala, M. (2022). Sign Language Recognition Using Deep Learning. 2022 4th International Conference on Cognitive Computing and Information Processing, CCIP 2022. https://doi.org/10.1109/CCIP57447.2022.10058655

[7] Tamam, Moh. B., Hozairi, H., Walid, M., Bernardo, J. F. A. (2023). Classification of Sign Language in Real Time Using Convolutional Neural Network. Applied In- formation System and Management (AISM), 6(1), 39–46. https://doi.org/10.15408/aism.v6i1.29820

[8] Neha Kawarkhe, M., Alhat, U., Wadaskar, P., Su- fiyan, S., Mohod, V., Ahmed Panjwani, M., Watane, H. N., Student, U. (n.d.). SIGN LANGUAGE RECOGNITION USING PYTHON. In International Research Journal of Modernization in Engineering Technology and Science www.irjmets.com @International Research Journal of Modernization in Engineering (Vol. 4900). www.irjmets.com

[9] Bronstein, M. M., Agapito, L., Rother, C. (2015). Preface. In Lecture Notes in Computer Science

(includ- ing subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 8927, p. VI). Springer Verlag. https://doi.org/10.1007/978-3-319- 16178-5

[10] Sabeenian, R. S., Sai Bharathwaj, S., Mohamed Aadhil, M. (2020). Sign language recognition using deep learning and computer vision. Journal of Advanced Research in Dynamical and Control Systems, 12(5 Special Issue), 964–968. https://doi.org/10.5373/JARDCS/V12SP5/20201842

[11] Pathan, R. K., Biswas, M., Yasmin, S., Khandaker, M. U., Salman, M., & Youssef, A. A. (2023). RETRACTED ARTICLE: Sign language recognition using the fusion of image and hand landmarks through multi-headed convolutional neural network. Scientific Reports, 13(1), 1-11. https://doi.org/10.1038/s41598-023-43852-x

[12] Liu, Y., Nand, P., Hossain, M.A. et al. Sign language recognition from digital videos using feature pyramid network with detection transformer. Multimed Tools Appl 82, 21673–21685 (2023). https://doi.org/10.1007/s11042-023-14646-0

[13] Deep, A. Litoriya, A. Ingole, V. Asare, S. M. Bhole and S. Pathak, "Realtime Sign Language Detection and Recognition," 2022 2nd Asian Conference on Innovation in Technology (ASIANCON), Ravet, India, 2022, pp. 1-4, https://doi.org/10.1109/ASIANCON55314.2022.9908995

[14] N. Rajasekhar, M. G. Yadav, C. Vedantam, K. Pellakuru and C. Navapete, "Sign Language Recognition using Machine Learning Algorithm," 2023 International Conference on Sustainable Computing and Smart Systems (ICSCSS), Coimbatore, India, 2023, pp. 303-306, https://doi.org/10.1109/ICSCSS57650.2023.10169820 . keywords: {Machine learning algorithms;Image color analysis;Gesture recognition;Speech recognition;Assistive technologies;Streaming media;Feature extraction;convolutional neural network;sign-recognition;hand movements},

[15] D. Van Hieu and S. Nitsuwat, "Image Preprocessing and Trajectory Feature Extraction based on Hidden Markov Models for Sign Language Recognition," 2008 Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, Phuket, Thailand, 2008, pp. 501-506, https://doi.org/10.1109/SNPD.2008.80