Leveraging Pre-Trained Neural Networks for Detecting American Sign Language

Nafiz Imtiaz Khan, Muhammad Hassnain, Md Raian Latif Nabil

1 Introduction & Motivation

Sign language is a vital mode of communication for individuals who are deaf or hard of hearing, relying on visual hand gestures to convey words, phrases and emotions [1]. Among the various sign languages, American Sign Language (ASL) stands out as one of the most extensively developed systems, complete with its own grammar rules, conventions, and a comprehensive lexicon of finger-spelling gestures representing alphabets and words[2]. Despite its richness, the utility of ASL is hindered by a critical barrier: it is primarily accessible to only those who have learned it.

This limitation creates a siginificant gap in communciation between the deaf community and the broader population, many of whom do not understand ASL. In everyday scenarios, such as shopping, healthcare or education, this communication barrier can lead to exclusion and dependency of interpreters, who are not always available. Consequently, fostering inclusivity requires bridging the gap to enable seamless two-way communication.

The inability of individuals unfamiliar with sign language to comprehend gestures is a persistent issue, that marginalizes the deaf community and restricts their access to essential services and social integration. For instance, a deaf individual might encounter challenges in a store when asking for assistance or face difficulties expressing themselves in an emergency when interpreters are unavailable.

To address this, we envision a system that automatically translates sign language into a text or speech in real time, enabling individuals to communicate without requiring prior knowledge of sign language. By leveraging advancements in computer vision and machine learning, such a system can recognize and interpret gestures from a video feed, effectively acting as a virtual interpreter. This innovation not only resolves immediate communication barriers but also promotes inclusivity by creating environments where individuals with hearing impairments can interact independently and confidently.

Our work focuses on designing and implementing this solution, starting with ASL gestures for alphabets to establish a scalable framework. By tackling this challenge, we aim to contribute to a more inclusive society, where technology bridges linguistic divides and empowers the deaf community.

2 Methodology

2.1 Data Gathering

In this project, we have used the MNIST ASL dataset [3] which consists of hand gesture images that represent 24 alphabets in the English language. Here, the letters J and Z are excluded, as their representation in ASL involves dynamic motion that cannot be captured in static images. It has 27455 training samples and 7172 test cases. Each image has 28 x 28 pixels in grayscale format with values ranging from 0-255. It holds a numerical value label of 0-25 (for example, 0 for, 3 corresponds to D etc.). The label distributions of the train and test image set is presented in Figure 1.

2.1.1 Model Selection

1. **CNN:** CNN [4] has always been preferable for image processing, object detection tasks, etc. We have used CNN as a benchmark model in our project.

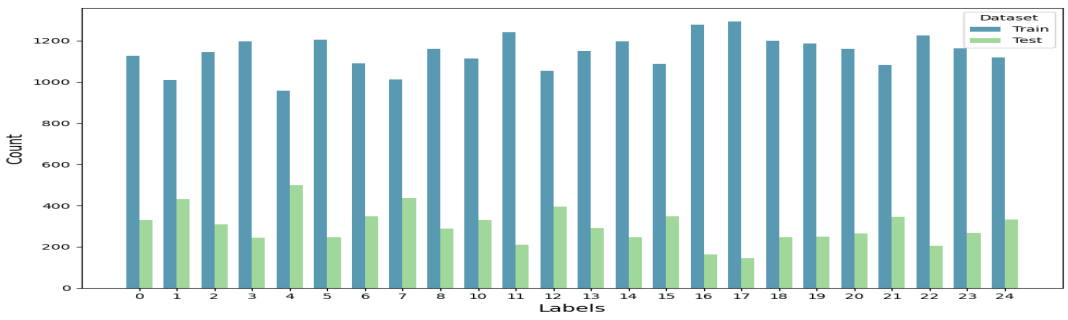


Figure 1: Distribution of class labels in both the training and test datasets

2. **ResNet50:** *ResNet50* [5] is a pre-trained CNN architecture with residual blocks that allow for skip connections on one or more layers. It resolves the degradation problem using residual blocks that allow the direct flow of information through the skip connections, which mitigates the vanishing gradient [6] problem.

3. **VGG16:** *VGG16* [7] is another pre-trained CNN model with 13 convolution layers and three fully connected layers, following the ReLU [8] activation function, established by AlexNet [9]. It was trained on imagenet dataset and gained 92.7% accuracy. It is also used in transfer learning which involves using a pre-trained *VGG16* model(trained on a larger dataset) as the base model and fine-tuning it for newer dataset.

4. **MobileNetV2:** *MobileNetV2* [10] is also a pre-trained CNN architecture that tends to perform well on mobile devices. It is based on an inverted residual structure where the residual connections are between the bottleneck layers. It uses a linear layer instead of ReLU to avoid losing information. This model is good for resource-constrained devices.

2.2 Developing Model Architectures

The architectural designs of the developed models are presented below. For the pre-trained models, we initially adopted the respective pre-trained model, then added custom layers and trained them for the specific sign language detection task.

1. **CNN:** The CNN model consists of a convolutional layer (conv1) followed by batch normalization (bn1) and a ReLU activation layer to extract features from the input image. After the convolutional block, the feature map is flattened and passed through three fully connected layers [11] (fc2, fc3, and fc4), with ReLU activations to the first two. The final output layer (fc4) is followed by a softmax function, which generates probabilities for the classes.

2. **ResNet50:** The model leverages a pre-trained *ResNet50* [5] backbone for feature extraction, with the original classification layer removed. The extracted features are passed through a custom classification head consisting of fully connected layers, batch normalization [12], ReLU activations, and dropout [13] for regularization. The output is passed through a softmax [14] function to generate class probabilities for the pre-defined labels.

3. **VGG-16:** The model utilizes a pre-trained *VGG16* backbone for feature extraction, removing its original classification layer. The extracted features are passed through a custom classification head, which includes a fully connected layer, batch normalization, ReLU activation, and dropout for regularization. The output is then passed through a softmax function to produce class probabilities for a multi-class classification task.

4. **MobileNetV2:** The model uses a pre-trained *MobileNetV2* as a feature extractor, where all layers are frozen to preserve the learned weights. The extracted features are passed through a custom classification head consisting of fully connected layers, batch normalization, ReLU activations, and dropout for regularization. Finally, the output is passed through a softmax function to generate probabilities for multi-class classification tasks.

2.2.1 Model Training

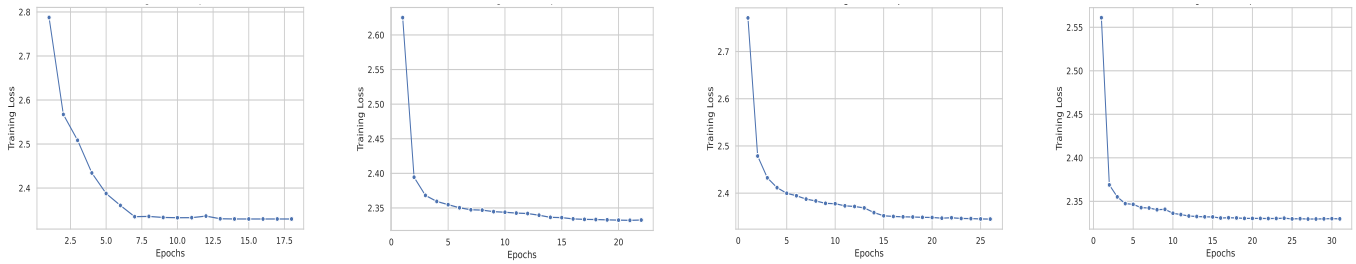
We have trained 4 models on the training data and evaluated it on both training and testing data on metrics- Accuracy, Precision, Recall, and F1-Score.

The neural network training process involves iterative optimization over a specific number of epochs (initially set as 200 for all the models), using the Adam optimizer [15] and cross-entropy [16] loss function to minimize loss. At each epoch, the model processes batches of training data, computes the loss, and updates the weights. Here, to maintain stability of the training, Gradient clipping [17] was applied. Training and testing accuracies are computed after each epoch to monitor performance, and the learning rate is adaptively adjusted by a learning scheduler based on accuracy improvements. Early stopping [18] is implemented to stop training if no improvement in testing accuracy is observed for a predefined number of epochs, *patient* (set to 5), ensuring efficient training and reducing overfitting. Throughout the process, training loss and accuracy metrics are saved for analysis.

Except for the first model, the last three models use transfer learning with pre-trained models, while the first model is a CNN model trained from scratch.

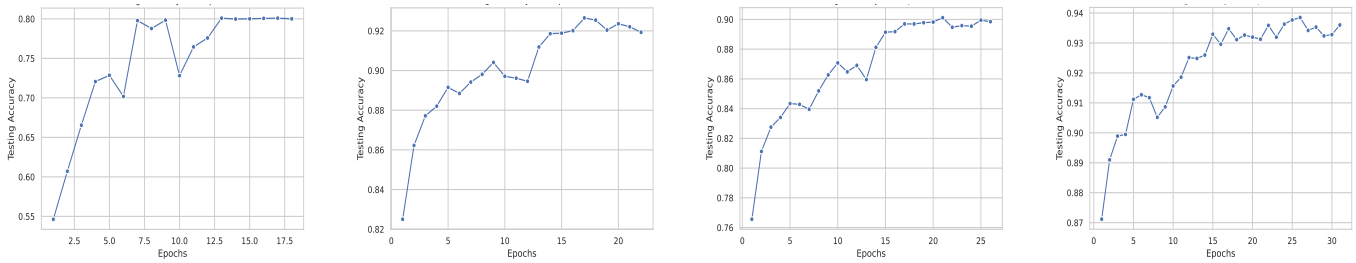
2.3 Development of the Real-Time Detection Application

We developed a system that performs real-time hand gesture detection through a streaming application. The application has a graphical user interface (GUI) for detecting ASL signs. It leverages OpenCV [19] library for video capture and image processing, Mediapipe [20] for hand landmark detection, and a custom machine learning model for prediction. It includes all functionalities, including video stream handling, real-time gesture detection, and GUI rendering. The GUI, built using PyQt6 [21], features a live video feed, prediction and confidence displays, and buttons for starting/stopping the capture and exiting the application. The system processes frames by detecting hands, cropping the hand region, resizing it to model input dimensions, and inferring gestures using the ModelAPI. The application ensures seamless user interaction and integrates dynamic updates for predictions, providing a robust framework for gesture-based applications.



(a) CNN (b) ResNet50 (c) VGG16 (d) MobileNetV2

Figure 2: Training loss of the models across epochs



(a) CNN (b) ResNet50 (c) VGG16 (d) MobileNetV2

Figure 3: Testing accuracy of the models across epochs

3 Results

Tale 1 shows the performance of the models on both training and testing sets. Training losses of the models across each epochs have been shown in Figure 2, while Test Accuracies of the models across the epochs have been shown in Figure 3.

From table 1, we can see that CNN model shows higher performance on the training data. CNN model acheieves nearly perfect results on the training data, on the other hand, the performance is on the lower side on the testing data. Thus, it can be inferred that this model has been overfitted on the training set. From the figure 2, we can see that the training loss becomes saturated after 8 epochs at first, then 13-14 epochs and it remains lower than 2.4. So thats why after 8 epochs, it jumps upto 80% accuracy and then drops down, but again at 13th epoch, it rides up to 80% and the testing accuracy curve becomes also flat and does not improve further.

ResNet50 gains close to perfect performance on the training data. It also scores higher in the testing data. So we can say the extremely high values suggest that the model has learned the training data very well, with a high ability to detect correct labels of the image sample. Precision and Recall scores are also at the higher side, which means the number of false positive prediction is pretty low. From the graph, we observe that *ResNet50* training loss saturates after 20 epochs, testing accuracy still changes after 20 epochs. *ResNet50* achieves highest accuracy at 17th epoch and then drops a little bit.

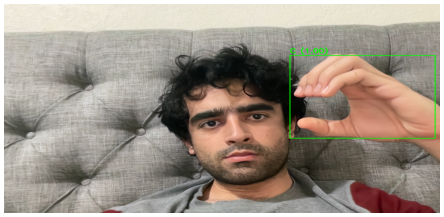
VGG16 also shows strong results, achieving more than 99% on each metrics. The model also performs at a high level on the test dataset with more than 91% scores on accuracy, precision, etc. It refers that it learns the pattern of image very well from the training images. Score on test dataset is also higher, but still slightly lower than the training score. From the graph, *VGG16* performs well to keep training loss at low and it goes flat after 20-25 epochs. This model gets the highest accuracy on 22nd epoch.

MobileNetV2 learns well with all training samples and so it scores 100% on each metrics, meaning that the model is classifying the training samples correctly and there is no false positive here. For the testing data, performance is little bit lower but it is still a high score ($>$ 94%). From the graph, *MobileNetV2* training loss stops reducing on 25-30 epochs. On the other hand, the testing accuracy does not remain flat.

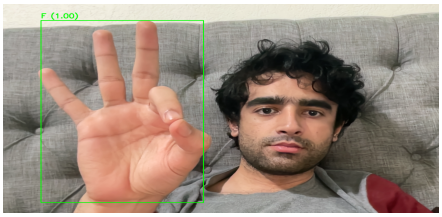
Among those models, *MobileNetV2* and *ResNet50* provide the best performance, their scores on training data are perfect, testing scores are comprehensive and higher and less possibility of false positive prediction. Their training loss is lesser than the other models and achieves great performance.

Table 1: Performance of the models on Train and Test dataset

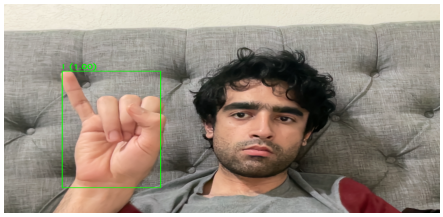
Model	**Train Dataset**				**Test Dataset**			
	Accuracy	**Precision**	**Recall**	**F1-Score**	**Accuracy**	**Precision**	**Recall**	**F1-Score**
CNN	99.78	99.79	99.79	99.78	80.06	79.57	78.48	77.90
ResNet50	99.99	99.99	99.99	99.99	95.04	94.81	94.97	94.84
VGG16	99.33	99.34	99.32	99.33	91.82	91.36	92.07	91.55
MobileNetV2	100	100	100	100	95.59	95.31	95.70	95.47



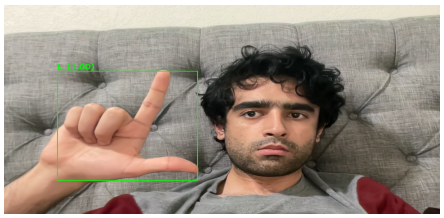
Character 'C'



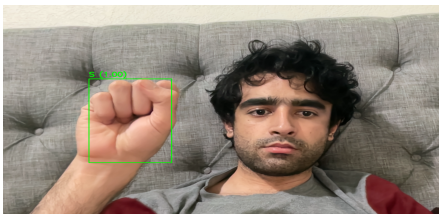
Character 'F'



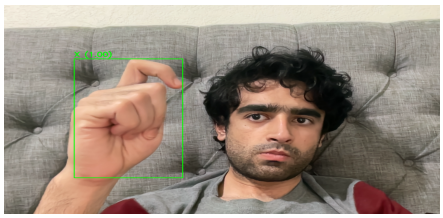
Character 'I'



Character 'L'



Character 'S'



Character 'X'

Figure 4: Screenshots from the real-time application showcasing the detection of different characters.

4 Discussion & Conclusion

The development and deployment of systems for real-time ASL recognition present several challenges. Our work addresses these challenges by integrating advanced ML techniques with real-time processing capabilities. By leveraging Mediapipe's hand detection framework[20] and a custom-trained deep learning model, we ensure precise detection and classification of hand gestures from live video feeds. We use PyQt for building an intuitive user interface further enhances the accessibility and usability of the system [22].

A key feature of our approach is the use of bounding box-based hand region extraction, combined with high-resolution cropping and resizing for model inference. This method mitigates the impact of noise from irrelevant parts of the frame, such as background objects or non-dominant hands. Additionally, by setting a high confidence threshold (95%) for predictions, we minimize the likelihood of erroneous classifications, ensuring reliable outputs. These advancements make the system suitable for real-world scenarios, such as assisting the deaf community in retail settings or emergency situations.

Despite these improvements, there are limitations to our current implementation. The system currently focuses on alphabet recognition, which restricts its practical use in recognizing more complex ASL words or sentences. Future work should explore the incorporation of temporal models, such as recurrent neural networks or transformers, to account for sequential gestures and enable full sentence recognition. Additionally, implementing domain adaptation techniques can improve robustness across varying lighting conditions and environments.

Another potential area for enhancement is user feedback integration. By allowing users to correct misclassifications in real-time, the system could iteratively improve its accuracy while simultaneously adapting to individual hand shapes and gesture styles. This adaptability could significantly broaden the user base and application scope of the system. learning

In conclusion, this work represents a significant step towards creating inclusive technologies that bridge the communication gap between the deaf community and the broader population. By combining state-of-the-art computer vision techniques with user-centric design, our ASL recognition system offers a scalable foundation for future innovations in assistive technology. While challenges remain, our findings highlight the potential for technology to foster inclusivity and empower marginalized communities through meaningful and impactful applications.

Team Members' Contributions

- **Nafiz**: Development and training of Models 1, 2, 3, and 4; bug fixing in model design; implementing model invokers and evaluator functions; writing the methodology section of the report.
- **Hassnain**: Idea conceptualization; developing the real-time detection app; bug fixing for model invocation; writing the introduction and discussion sections of the report.
- **Nabil**: Development and training of Models 5 and 6; bug fixing in model design and invokers; writing the results section of the report. *(Due to space limitation results from model 5 and 6 were not included in the report)*

Data & Code Availability

The project's data and code are publicly available on GitHub [23] [1] and the final version, including model weights, is archived on Zenodo [24] [2].

[1] https://github.com/muhammad-hassnain/ASL-Recognition-System
[2] https://zenodo.org/records/14347694

References

[1] Razieh Rastgoo, Kourosh Kiani, and Sergio Escalera. Sign language recognition: A deep survey. *Expert Systems with Applications*, 164:113794, 2021.

[2] Clayton Valli and Ceil Lucas. *Linguistics of American Sign Language: An Introduction*. Gallaudet University Press, 2000.

[3] Datamunge. Sign language mnist. https://www.kaggle.com/datasets/datamunge/sign-language-mnist, 2017. Accessed: 2024-12-13.

[4] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[6] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[7] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[8] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[10] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. https://www.deeplearningbook.org.

[12] Sergey Ioffe. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[13] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. In *Journal of Machine Learning Research*, volume 15, pages 1929–1958, 2014.

[14] A. Joulin, T. Mikolov, and Q.V. Le. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2017. URL https://arxiv.org/abs/1607.01759.

[15] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[17] R Pascanu. On the difficulty of training recurrent neural networks. *arXiv preprint arXiv:1211.5063*, 2013.

[18] Lutz Prechelt. Automatic early stopping using cross validation: quantifying the criteria. *Neural networks*, 11(4): 761–767, 1998.

[19] Gary Bradski. The opencv library. https://opencv.org/, 2000. Accessed: 2024-12-13.

[20] Google Research. Mediapipe hands: Real-time hand detection and tracking, 2023. Available at: https://mediapipe.dev/hands.

[21] Riverbank Computing Limited. Pyqt6: Python bindings for the qt6 toolkit. https://www.riverbankcomputing.com/software/pyqt/intro, 2021. Accessed: 2024-12-13.

[22] Riverbank Computing Limited. Pyqt: Python bindings for qt, 2023. Available at: https://www.riverbankcomputing.com/software/pyqt/intro.

[23] Muhammad Hassnain, Nafiz Imtiaz Khan, and Md Raian Latif Nabil. Asl recognition system. https://github.com/muhammad-hassnain/ASL-Recognition-System, 2024. Accessed: 2024-12-08.

[24] Nafiz Imtiaz Khan. Record title, 2018. URL https://zenodo.org/records/14347694.