CS 584 - F24

Sign Language Detection Phase III

Team 12
Vignesh Ram Ramesh Kutti (A20548747)
Vishnu Priyan Sellam Shanmugavel (A20561323)
Aravind Balaji Srinivasan (A20563386)
Kavin Raj Karuppusamy Ramasamy (A20564249)
Kishan Murali (A20601493)

Project Goal

Task (T)

- Recognizing sign language gestures using neural networks.

Experience (E)

- Measured by the percentage of gestures correctly classified by the model.

Performance (P)

- Utilized a database of labeled images of sign language gestures for training the model.

Steps Involved

- **Data Collection:** Gather a diverse dataset of sign language gestures from different source.
- **Data Cleaning:** Preprocess the data to remove noise and inconsistencies.
- **Feature Extraction:** Extract key features like hand shapes, orientations, and movements, using neural networks techniques.
- **Model Training:** Train a neural network model on the preprocessed data to recognize and classify sign language gestures.
- **Model Validation:** Evaluate the model's performance using a validation dataset.
- **Model Testing:** Test the model on new data to ensure generalization.
- **Real-Time Recognition:** Implement the trained model to recognize and translate gestures in real-time.

Phase I (completed)

- **Sources:** Gather data from diverse sources, including existing datasets, crowdsourcing platforms, and in-house recordings.
- **Diversity:** Ensure the dataset includes a wide range of signers, backgrounds, and lighting conditions.
- **Quantity:** Aim for a large dataset to provide sufficient training data.
- **Noise Removal:** Remove unwanted noise, artifacts, and distortions from the data.
- **Inconsistency Handling:** Address inconsistencies in data formatting, labeling, and gesture performance.
- **Normalization:** Standardize the data to a common scale for consistent input to the neural network.

Datasets for the Project

- Dataset 1: ASL Dataset
- Dataset 2: Interpret Sign Language with Deep Learning
- Dataset 3: Sign Language MNIST
- Dataset 4: Sign Language Detection Using Images

Phase II Steps

Feature Extraction

- **Deep Learning Techniques:** Explore deep learning techniques like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) to automatically extract relevant features from the images or videos.

Model Training

- **Neural Network Architecture:** Experiment with different neural network architectures, such as CNNs, RNNs, or hybrid models, to find the most suitable approach for your task.
- **Hyperparameter Tuning:** Optimize hyperparameters (e.g., learning rate, batch size, number of epochs) to improve model performance.
- **Transfer Learning:** Consider using pre-trained models (e.g., from image classification tasks) as a starting point to accelerate training and potentially improve accuracy.

Convolutional Neural Networks(CNN)

- **Captures Spatial Hierarchies**: CNNs learn filters that detect patterns like edges, textures, and shapes in a hierarchical manner, crucial for tasks like object detection and classification.
- **Focuses on Local Details**: CNNs use local receptive fields to focus on small image regions, improving pattern recognition.
- **Efficient with High-Dimensional Data**: CNNs process images in smaller sections, reducing computational complexity and handling high-dimensional data efficiently.
- **Optimized for Image Tasks**: CNNs excel in image-based tasks like classification, object detection, and segmentation due to their structure.
- **Convolution:** Usually, images have 3 channels for RGB which the filter (kernel) convolves to extract features like edges and patterns effectively in CNNs

CNN Example for Image Classification

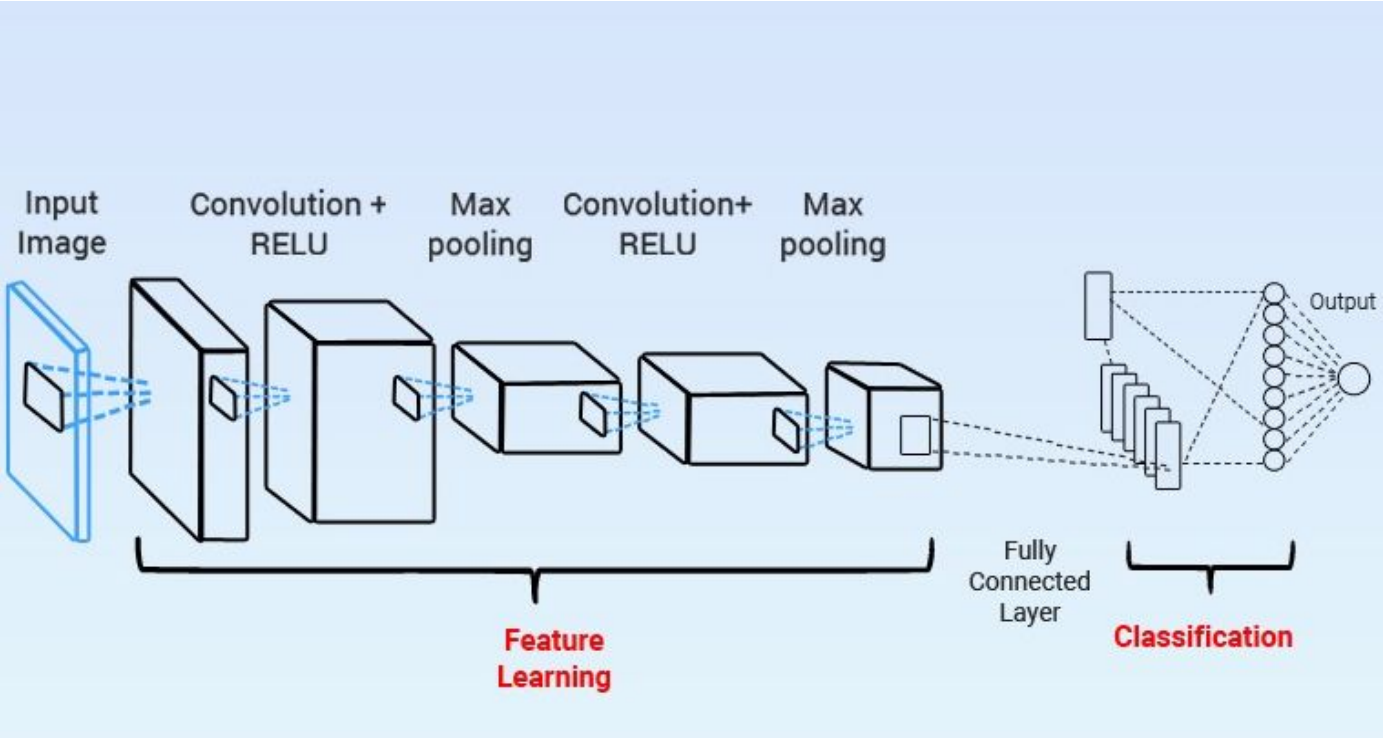
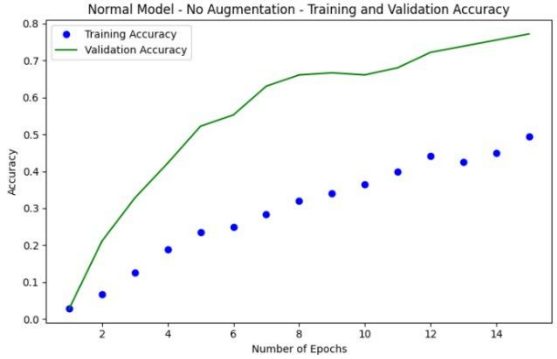


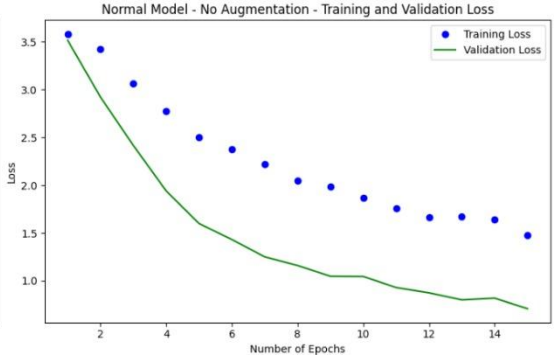
Image Classification: An Overview Link

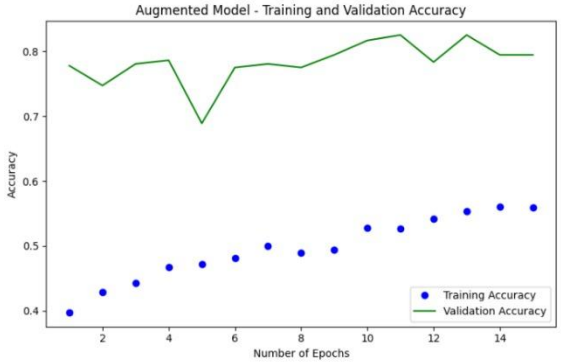
Model 1: Standard Convolutional Neural Network (CNN)

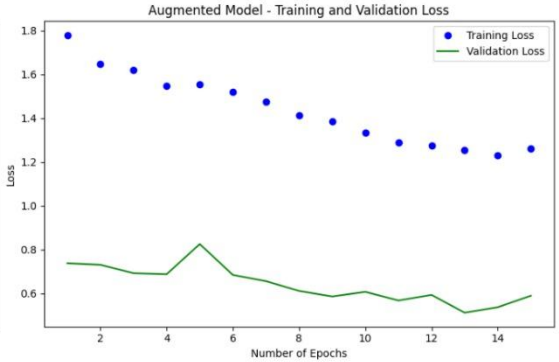
Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 62, 62, 32)	896
max_pooling2d (MaxPooling2D)	(None, 31, 31, 32)	0
conv2d_1 (Conv2D)	(None, 29, 29, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_2 (Conv2D)	(None, 12, 12, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 128)	0
flatten (Flatten)	(None, 4608)	0
dense (Dense)	(None, 128)	589,952
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 36)	4,644







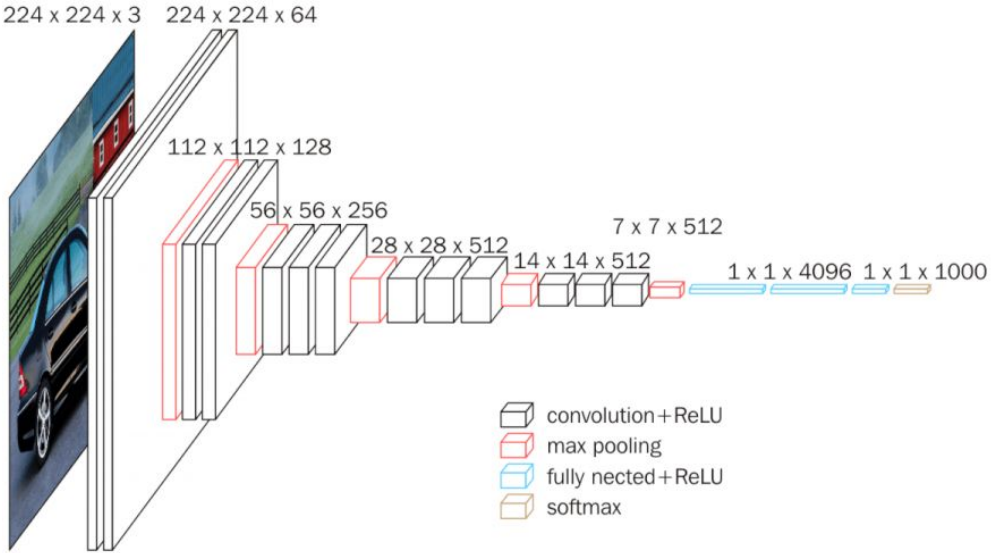


Why Data Augmentation?

- **Improves Model Generalization**: Data augmentation increases the variety of training data, helping models generalize better to unseen data.
- **Reduces Overfitting**: By introducing variations like rotations, flips, and zooms, augmentation prevents models from memorizing training data, reducing overfitting.
- **Enhances Robustness**: Augmentation introduces noise and variability, making the model more robust to real-world variations in input data.
- **Maximizes Small Datasets**: It artificially expands limited datasets, allowing better model performance when data is scarce.

Model 2: Pre-trained network VGG-16

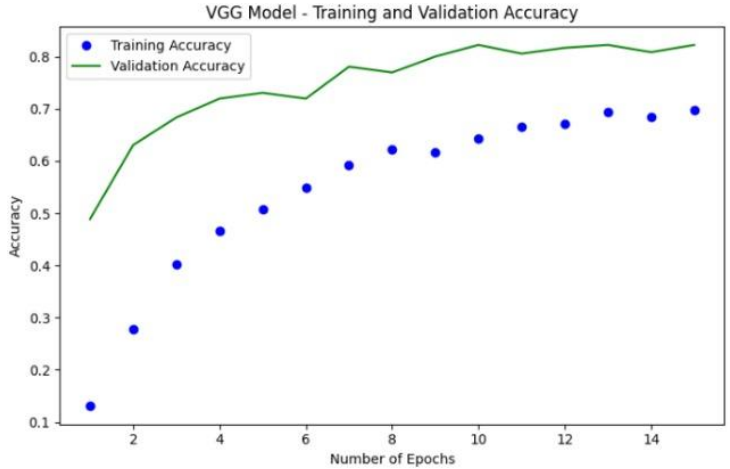
- **Deep Architecture**: VGG-16 is a deep network with 16 layers, enabling it to capture complex hierarchical features from images.
- **Small Convolutions**: Uses 3x3 convolution filters throughout, focusing on small receptive fields for precise feature extraction.
- **High Accuracy**: Known for achieving top performance on image classification tasks, particularly in large-scale datasets like ImageNet.
- **Transfer Learning**: Widely used for transfer learning due to its pre-trained weights, allowing it to adapt quickly to new tasks with minimal fine-tuning.

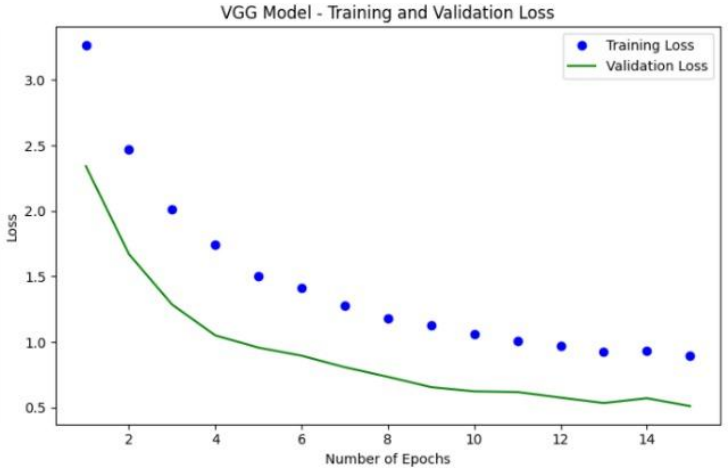


VGGNet-16 Architecture - [source]

Pre-trained network VGG-16 Metrics

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 2, 2, 512)	14,714,688
flatten_1 (Flatten)	(None, 2048)	0
dense_2 (Dense)	(None, 256)	524,544
dropout_1 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 36)	9,252



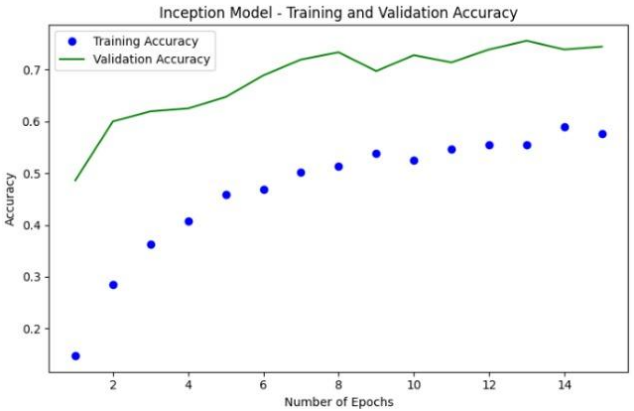


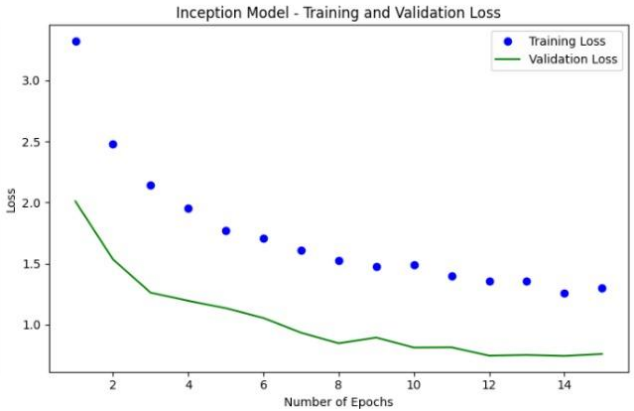
Pre-trained network Inception V3

- **Efficient Architecture:** Inception V3 uses a deep network with a focus on computational efficiency, employing factorize convolutions to reduce computational cost.
- **Inception Modules:** Utilizes inception modules that allow the network to capture multi-scale features in a single layer by applying multiple filter sizes simultaneously.
- **Batch Normalization:** Includes extensive use of batch normalization, which helps to stabilize and accelerate training by normalizing intermediate layers.
- **Transfer Learning:** Frequently used for transfer learning due to pre-trained weights, enabling effective fine-tuning for various image-related tasks.

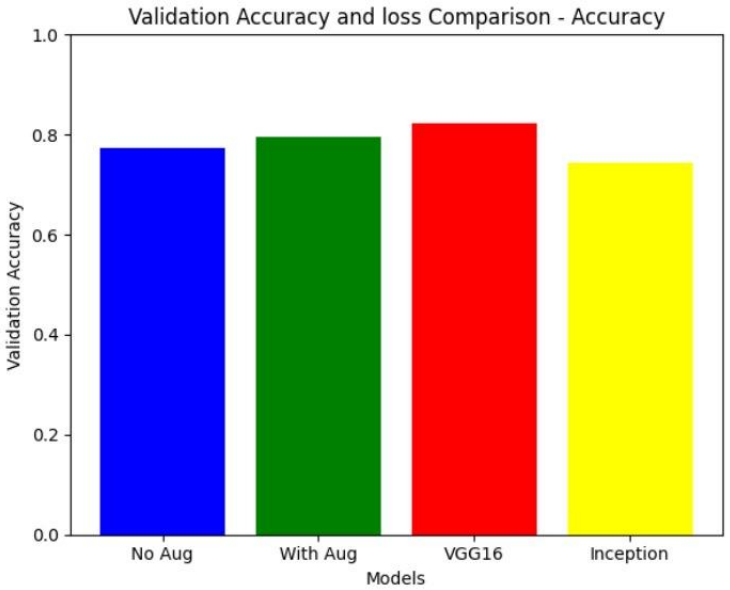
Pre-trained network Inception V3

Layer (type)	Output Shape	Param #
inception_v3 (Functional)	(None, 1, 1, 2048)	21,802,784
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0
dense_4 (Dense)	(None, 256)	524,544
dropout_2 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 36)	9,252





Model Comparison



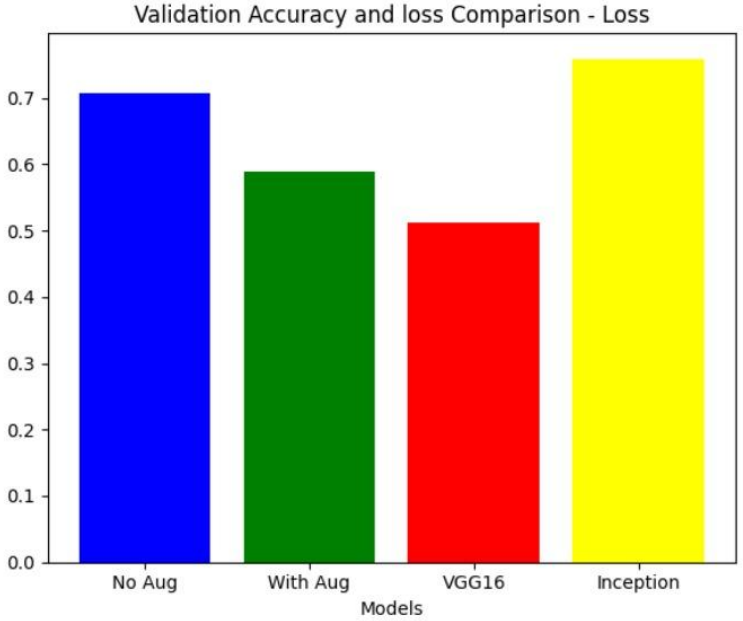


Table showing Accuracy and Loss values for different models.

Model	Accuracy	Loss
Standard CNN	78%	0.7
Standard CNN with Data Augmentation	80%	0.6
Inception V3	70%	0.75
Visual Geometry Group (VGG-16)	82%	0.5

We chose the VGG-16 model for its highest accuracy (82%) and lower loss, providing the best performance for our recognition task.

Challenges and Feasibility

- Building neural networks for accurate gesture recognition.
- Overfitting of data was evident with normal CNN models. We overcame this challenge by introducing data augmentation in the dataset for training the model.
- When training Inception V3 pre-trained model we observed less accuracy scores and high losses since we did not feed it sufficient data to train and perform as expected. This is due to the limited computation power of our GPUs.
- When augmenting data, when we rotate the images by just 20 degrees, the meaning of the new image is changed as it might represent a new character. So we avoided using rotation_range and shear while augmenting the data.

Issues that needs to be resolved

- **Inability to Detect Gestures**: The current model, which relies solely on static images, cannot detect dynamic gestures like "J" and "Z," as these require real-time motion capture to track hand movement.
- **Confusion with digits and alphabets:** some of the numbers and alphabets have similar sign language, so the model struggles to classify them properly. So we intend to just classify alphabets.

Link to our model implementation

Github Link: https://github.com/vishnu32510/sign_language

Phase III

1. Model Validation & Testing

- **Performance Metrics:** Use validation and test datasets to evaluate metrics like Accuracy, Precision, Recall, and F1-Score.
- **Generalization Check:** Ensure the model performs well on unseen data. Utilize confusion matrices and ROC curves for analysis.
- **Refinement**: Adjust the model based on results, and document failure cases for improvement.
- **Model Selection:** Choose the most suitable model architecture based on performance metrics, balancing accuracy and computational efficiency.

Phase III

2. Motion-Specific Gestures (J & Z)

Challenge: J and Z require motion detection for accurate recognition.

Approach:

- Use video sequences to capture the full motion.
- Incorporate multiple frames to represent the gesture's trajectory.
- Track start and end positions to distinguish the dynamic movement of J and Z.

Phase III

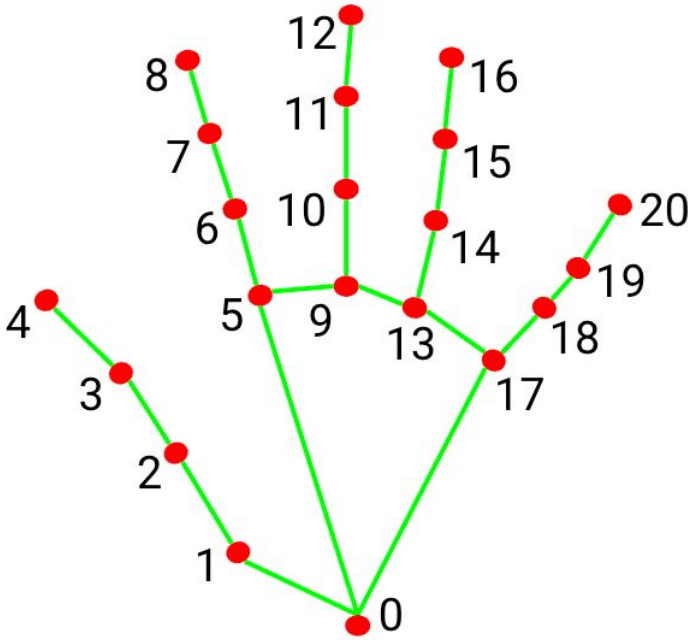
3. Real-Time Recognition

- **Deployment**: Implement the trained model for live gesture recognition.
- **Optimization:** Focus on low latency and robustness across diverse environments.
- **Goal**: Achieve accurate, real-time translation of sign language gestures into text or speech.

MediaPipe

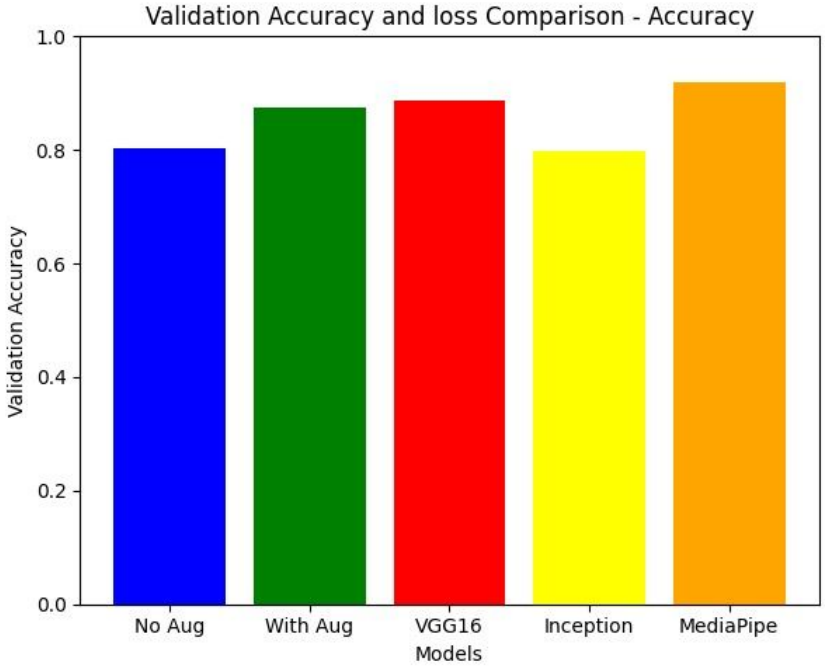
- **Versatile Framework**: MediaPipe offers pre-built solutions for tasks like hand tracking and pose estimation.
- **Real-Time Performance**: Optimized for real-time video and image processing.
- **Cross-Platform**: Supports Python, C++, and JavaScript across various devices.
- **Hardware Acceleration**: Leverages GPU for efficient processing.
- **Customizable**: Easily integrates with custom machine learning models.

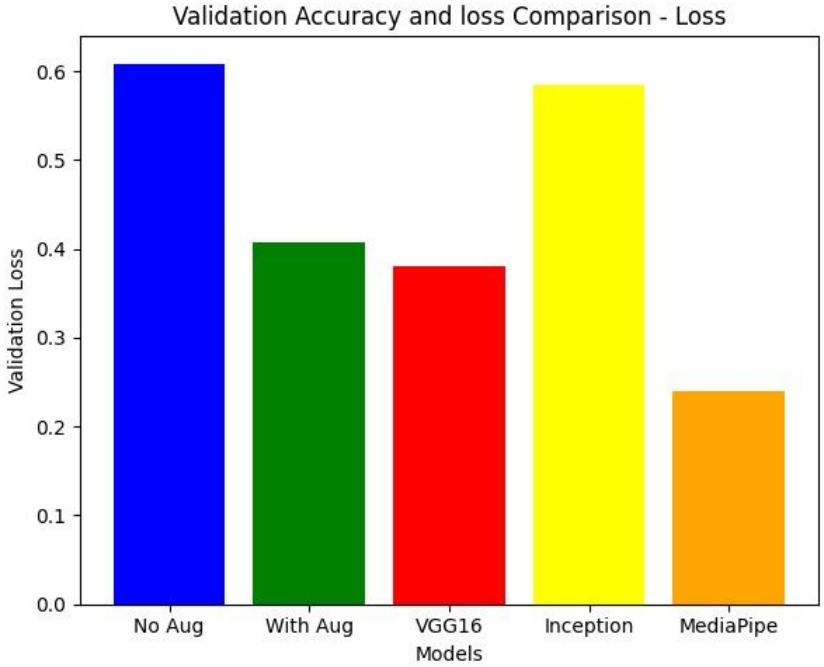
MediaPipe



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

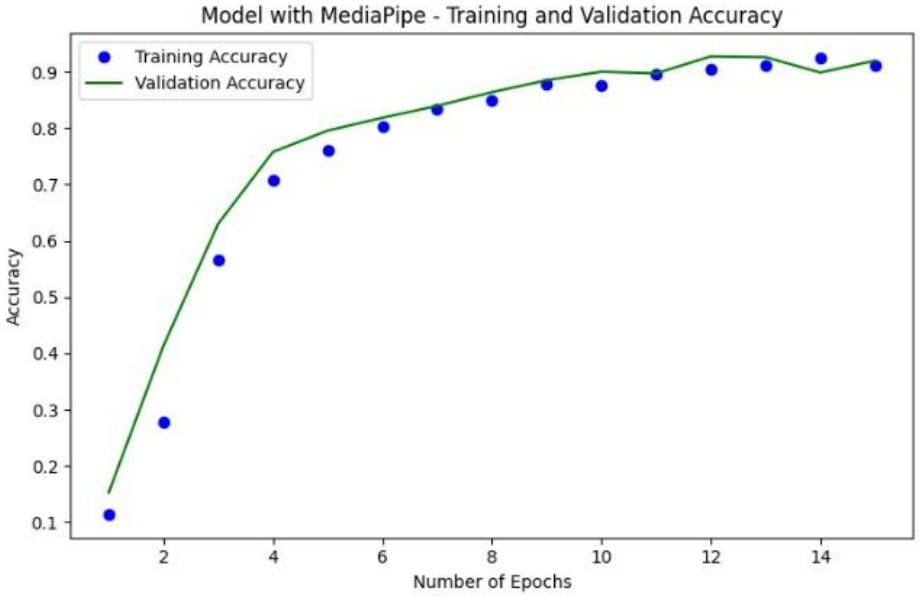
Model Selection

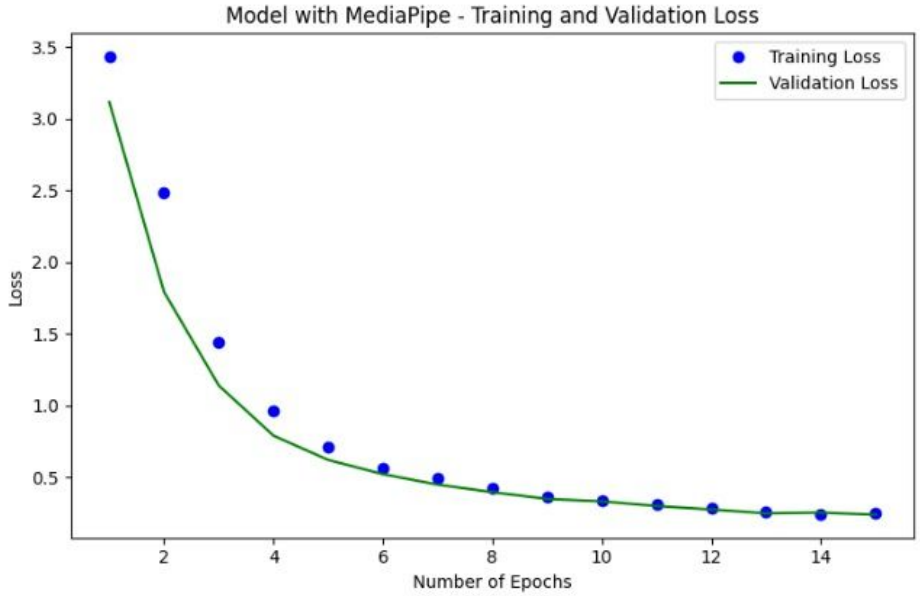




Since MediaPipe performed better than other approaches with very less loss, We implemented MediaPipe for detection of signs in Real-time.

MediaPipe Performance





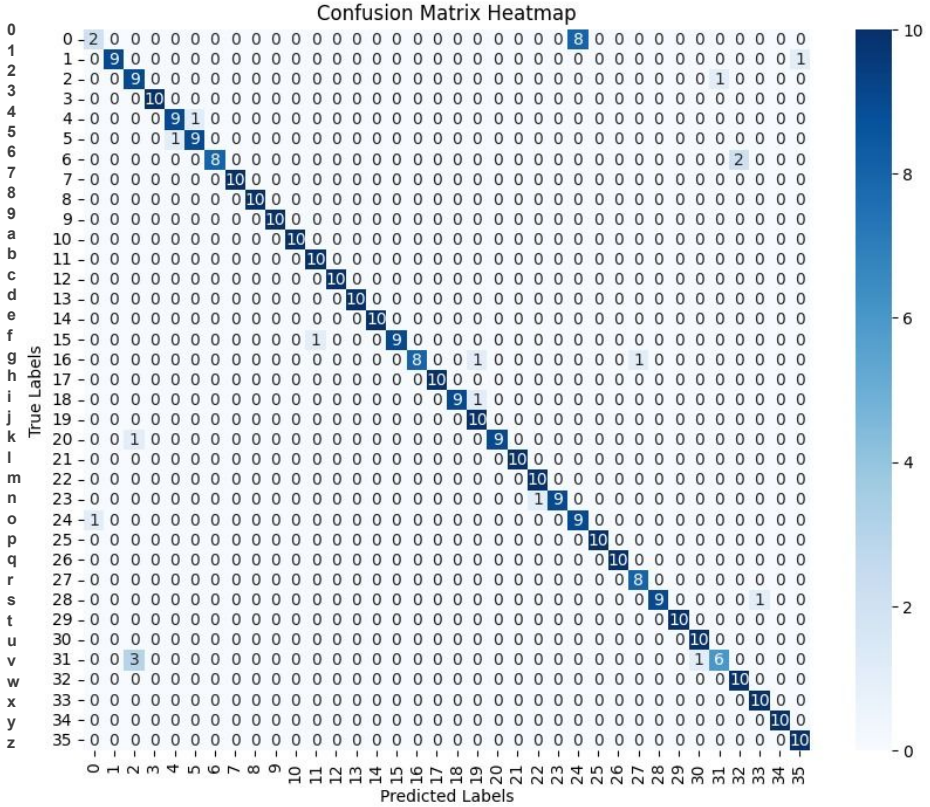
MediaPipe Metrics(Class Wise)

ILLINOIS TECH

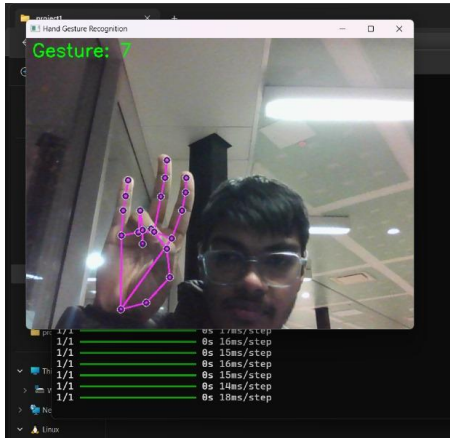
Class Label	Precision	Recall	F1-Score	Support
0	0.67	0.2	0.31	10
1	1	0.9	0.95	10
2	0.69	0.9	0.78	10
3	1	1	1	10
4	0.9	0.9	0.9	10
5	0.9	0.9	0.9	10
6	1	0.8	0.89	10
7	1	1	1	10
8	1	1	1	10
9	1	1	1	10
10	1	1	1	10
11	0.91	1	0.95	10
12	1	1	1	10
13	1	1	1	10
14	1	1	1	10
15	1	0.9	0.95	10

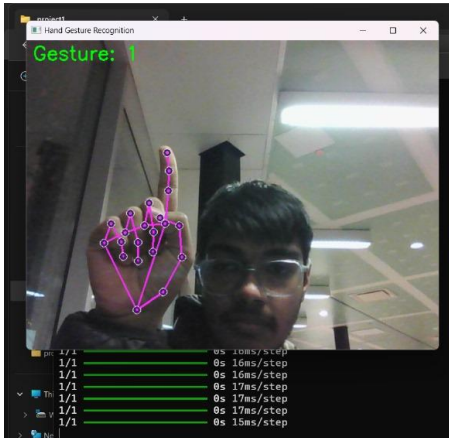
Class Label	Precision	Recall	F1-Score	Support
18	1	0.9	0.95	10
19	0.83	1	0.91	10
20	1	0.9	0.95	10
21	1	1	1	10
22	0.91	1	0.95	10
23	1	0.9	0.95	10
24	0.53	0.9	0.67	10
25	1	1	1	10
26	1	1	1	10
27	0.89	1	0.94	8
28	1	0.9	0.95	10
29	1	1	1	10
30	0.91	1	0.95	10
31	0.86	0.6	0.71	10
32	0.83	1	0.91	10
33	0.91	1	0.95	10
34	1	1	1	10
35	0.91	1	0.95	10

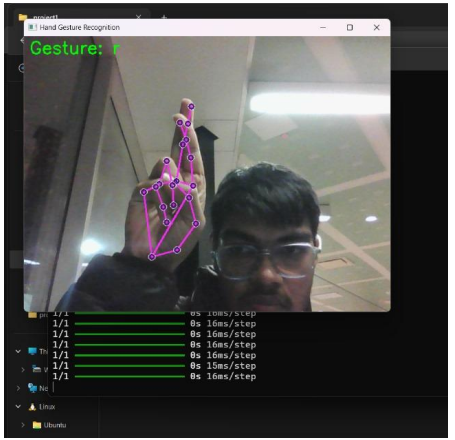
MediaPipe Confusion Matrix

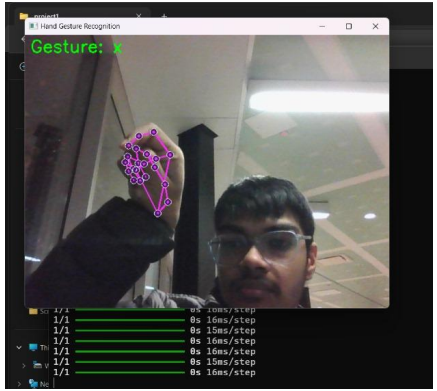


Realtime Results

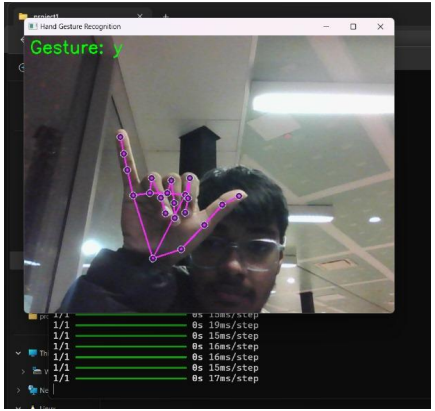








[Demo Link]



Conclusion

- **Robust Validation**: Achieved reliable classification with strong performance metrics.
- **Real-Time Integration**: MediaPipe enabled efficient, low-latency gesture recognition.
- **Practical Potential**: Demonstrated effective sign language translation for real-world use.
- **Efficient Design**: Balanced model performance with computational limitations.
- **Future Impact**: Paves the way for accessible technologies and inclusive communication.

Future Scope

- **Improved Gesture Recognition**: Expanding the model to handle dynamic gestures and more complex sign sequences.
- **Utilizing Transformers**: Leveraging transformer architectures to significantly enhance recognition accuracy and model efficiency.
- **Multilingual Support**: Adapting the system for different sign languages globally.
- **Enhanced Real-Time Capabilities**: Optimizing latency and performance for seamless integration in diverse environments.
- **Integration with Devices**: Developing applications for smartphones, AR/VR, and IoT devices to improve accessibility.

Applications & Future Scope

- **Real-time Sign Language Interpretation**

  Enabling communication between deaf and hearing individuals by converting sign language into spoken language or text.

- **Accessible Technology**

  Developing accessible interfaces for devices and software, allowing deaf individuals to interact with technology using sign language.

- **Sign Language Education**

  Creating interactive learning tools and platforms for sign language education, enhancing accessibility and promoting inclusivity.

Problems Faced

- **Similarity in Symbols**: Confusion between gestures with similar appearances, such as "0" and the letter "O."
- **Dynamic Gesture Recognition**: Difficulty in detecting dynamic hand gestures like "J" and "Z," which require motion tracking.
- **Real-Time Performance**: The model struggled with real-time detection, leading to the integration of MediaPipe for enhanced efficiency.
- **Limited Computational Resources**: Inability to utilize advanced models like transformers due to hardware constraints.
- **Dataset Variability**: Challenges in achieving consistent performance across diverse lighting, backgrounds, and signer variations.

References

1. Camgoz et al., 2018: Neural Sign Language Translation. CVPR 2018 Paper
2. NeurIPS 2022: Multimodal Sign Language Translation. NeurIPS 2022 Paper
3. DeepASLR: A CNN based human computer interface for American Sign Language recognition for hearing-impaired individuals - DeepASLR Paper
4. MediaPipe: Library from Google - Documentation

Thank You! ;)