Project Title - Sign Language Recognition

GUIDED BY

DR. JEETHU V. DEVASIA

SUBMITTED BY

22BCE20346 - Valavala Lakshmi Nagendra

22BCE7426: Shaik Nehaal Khasim

21BCE8210 - Pasurla Bharath Simha Reddy

SCHOOL OF COMPUTER SCIENCE & ENGINEERING, VIT-AP UNIVERSITY

Fall Sem 2024-2025

Chapter 1: Introduction

Introduction

Sign Language Recognition (SLR) is a technology that helps translate sign language into text or speech. American Sign Language (ASL) is widely used by deaf and hard-of-hearing communities in the U.S. and Canada. ASL uses hand shapes and movements to represent letters and words.

Motivation & Relevance

The main motivation for this project is to make communication easier for deaf and hard-of-hearing people. Currently, communicating with ASL can require a sign language interpreter, which might not always be available.

By creating an automated system that can recognize ASL signs, we can help bridge this communication gap. This technology is important because it can improve accessibility in various situations, such as in public places, educational settings, and online platforms. Making communication more accessible supports inclusivity and ensures that everyone can participate more fully in society.

Objective

The goal of this project is to build a system that can accurately recognize static ASL signs using deep learning methods. We are using three models: **VGG16, ResNet50** and **SignNet**.

- **VGG16** is widely appreciated for its simplicity and uniform architecture, with 16 layers arranged in a sequential manner, enabling efficient extraction of complex spatial features from images while maintaining computational feasibility.

- **ResNet50** (Residual Network) is known for its ability to train very deep networks effectively by using residual connections, which help mitigate the vanishing gradient problem.

- **SignNet** is specifically designed for sign language recognition, leveraging spatiotemporal information for accurate hand gesture detection.

The system will take images of hand signs and determine what each sign represents. Our aim is to create a model that is both accurate and reliable, helping to advance the field of

sign language recognition and potentially leading to more tools that can assist with communication.

Problem Statement

Recognizing ASL signs from images is challenging for several reasons. First, many different hand shapes and positions exist, making it difficult to distinguish between similar signs. Second, environmental factors like lighting, background, and variations in the way signs are performed further complicate recognition.

Even with advanced technology, accurately classifying these signs is a complex problem. This project aims to address these challenges by using **ResNet50, VGG16** and **SignNet**.

- **VGG16**'s deep, sequential architecture with small convolutional filters enables it to capture detailed spatial hierarchies, making it effective for recognizing fine-grained features in static images.

- **ResNet50**'s residual connections allow for more accurate feature extraction, making it well-suited for static image classification tasks.

- **SignNet**, with its ability to capture spatiotemporal dependencies, enhances recognition by understanding contextual information present in sequences of gestures.

The goal is to develop a model that can handle these variations effectively and reliably identify static ASL signs.

Chapter 2: Literature review

Research Papers

1) **"A Comprehensive Review of Sign Language Recognition: Different Types, Modalities, and Datasets" (2022)** This paper provides an overview of SLR systems, focusing on vision-based and sensor-based methods. It highlights the challenges in feature extraction, noise interference, and environmental factors. The authors emphasize the importance of preprocessing techniques to improve accuracy in real-time applications. The paper discusses the latest methods like DWT, SIFT, and PCA for feature extraction.

2) **"Sign Language Recognition Using Multiple Kernel Learning: A Case Study of Pakistan Sign Language" (2021)**
This study applied multiple kernel learning to recognize Pakistan Sign Language. The system uses features from video sequences and outperformed traditional approaches like SVM and CNN in handling sign variations. The use of multiple kernels improves model generalization, though it requires large computational resources.

3) **"Vision-Based Hand Gesture Recognition Using Deep Learning for the Interpretation of Sign Language" (2021)**
This research utilizes deep learning models to identify hand gestures in real-time video feeds. A combination of CNN and RNN was used to capture temporal hand movements. The model's accuracy was high, though the requirement of high-quality input data limited its performance in low-light conditions.

4) **"American Sign Language Identification Using Hand Trackpoint Analysis" (2021)**
The authors proposed a system based on trackpoint analysis for detecting hand positions and movements in ASL. Using a CNN, the system achieved promising

results in recognizing signs from a video feed. However, the model struggled with signs involving facial expressions.

5) "Indian Sign Language Recognition System Using SURF with SVM and CNN" (2022)

This paper explored the use of SURF (Speeded Up Robust Features) along with SVM and CNN models to classify hand gestures in Indian Sign Language. It improved accuracy significantly by combining these methods, although the computational complexity was high.

6) "Isolated Sign Language Recognition Using 3D Convolutional Networks" (2022)

This research implemented 3D Convolutional Networks for isolated

SLR using motion history images. The model proved effective in recognizing dynamic signs with high accuracy. However, the system was less effective with continuous sign sequences.

7) "Applying Hybrid Deep Neural Networks for Sign Language Recognition in COVID-19 Patients" (2022)

This study focused on hybrid deep neural networks to help communicate with deaf COVID-19 patients using sign language. The model utilized both CNNs and RNNs to recognize signs from video sequences. It highlighted the advantage of using transfer learning to enhance accuracy.

8) "Real-Time Hand Sign Language Recognition Using Deep Networks and SVD" (2021)

The paper introduced a real-time hand sign recognition system using a combination of CNN and singular value decomposition (SVD) for dimensionality reduction. The model excelled in speed and performance but required high-quality cameras for best results.

9) "Application of Transfer Learning to Sign Language Recognition using an Inflated 3D Deep Convolutional Neural Network" (2021) This paper explored transfer learning techniques combined with 3D CNN for sign language

recognition. The use of pre-trained models drastically reduced training time and improved accuracy, but the dataset size remained a limiting factor for performance.

10) **"Deep Learning Technology to Recognize American Sign Language Alphabet" (2023)**
This paper introduces a CNN-based system for ASL alphabet recognition. The model uses image processing techniques to recognize hand gestures, relying on high-resolution camera images. The dataset consists of 104,000 images representing 26 ASL letters. The model achieved a 99% accuracy rate across three different datasets. Advantages include high accuracy, but it struggles with dynamic signs and complex backgrounds.

Comparison Table

Sl. No	Title of the Paper	Methodology	Datasets Used	Performance Metrics	Advantages	Disadvantages
1	A Comprehensive Review of Sign Language Recognition (2022)	Review of SLR methods	Various public datasets	N/A	Comprehensive review of methods	Limited original experimentation

2	Sign Language Recognition Using Multiple Kernel Learning (2021)	Multiple kernel learning	Custom Pakistan SL dataset	95% accuracy	High generalization	Computationally expensive
3	Vision-Based Hand Gesture Recognition Using Deep Learning (2021)	CNN + RNN	Custom dataset	92% accuracy	Real-time application	Low performance in poor lighting
4	American Sign Language Identification Using Hand Trackpoint Analysis (2021)	CNN	ASL Dataset	93% accuracy	Accurate hand movement detection	Struggles with facial expressions
5	Indian Sign Language Recognition Using SURF with SVM and CNN (2022)	SURF + SVM + CNN	Custom ISL dataset	89% accuracy	High feature detection	High complexity
6	Isolated Sign Language Recognition Using 3D Convolutional Networks (2022)	3D CNN	Custom dataset	94% accuracy	Effective for dynamic signs	Poor with continuous signs
7	Hybrid Deep Neural Networks for Sign Language in COVID-19 Patients (2022)	CNN + RNN + Transfer learning	Custom medical dataset	91% accuracy	Combines multiple networks	Dependent on high-quality video
8	Real-Time Hand Sign Language Recognition Using SVD (2021)	CNN + SVD	Custom hand dataset	90% accuracy	Fast processing	Requires highquality cameras

9	Transfer Learning for Sign Language Using 3D CNN (2021)	3D CNN + Transfer Learning	Public datasets	92% accuracy	Reduced training time	Dataset limitations
10	Deep Learning Technology to Recognize ASL Alphabet	CNN	104000 ASL images	99% accuracy	High accuracy, handles various lighting conditions	Limited to static signs, struggles with dynamic gestures

References: (Citations)

1) Madhiarasan, D. M., Roy, P., & Pratim, P. (2022). A comprehensive review of sign language recognition: Different types, modalities, and datasets. arXiv preprint arXiv:2204.03328.

2) Shah, F., Shah, M. S., Akram, W., Manzoor, A., Mahmoud, R. O., & Abdelminaam, D. S. (2021). Sign language recognition using multiple kernel learning: A case study of Pakistan sign language. Ieee Access, 9, 67548-67558.

3) Sharma, S., & Singh, S. (2021). Vision-based hand gesture recognition using deep learning for the interpretation of sign language. Expert Systems with Applications, 182, 115657.

4) Bajaj, Y., & Malhotra, P. (2022). American sign language identification using hand trackpoint analysis. In International Conference on Innovative Computing and Communications: Proceedings of ICICC 2021, Volume 1 (pp. 159-171). Springer Singapore.

5) Katoch, S., Singh, V., & Tiwary, U. S. (2022). Indian Sign Language recognition system using SURF with SVM and CNN. Array, 14, 100141.

6) Huang, J., Zhou, W., Li, H., & Li, W. (2015, June). Sign language recognition using 3d convolutional neural networks. In 2015 IEEE international conference on multimedia and expo (ICME) (pp. 1-6). IEEE.

7) Venugopalan, A., & Reghunadhan, R. (2023). Applying hybrid deep neural network for the recognition of sign language words used by the deaf Covid-19 patients. Arabian Journal for Science and Engineering, 48(2), 1349-1362.

8) Rastgoo, R., Kiani, K., & Escalera, S. (2022). Real-time isolated hand sign language recognition using deep networks and SVD. Journal of Ambient Intelligence and Humanized Computing, 13(1), 591-611.

9) Töngi, R. (2021). Application of transfer learning to sign language recognition using an inflated 3D deep convolutional neural network. arXiv preprint arXiv:2103.05111.

10) Alsharif, B., Altaher, A. S., Altaher, A., Ilyas, M., & Alalwany, E. (2023). Deep learning technology to recognize American sign language alphabet. Sensors, 23(18), 7970.

Chapter 3: Methods

1. Model Name: VGG16

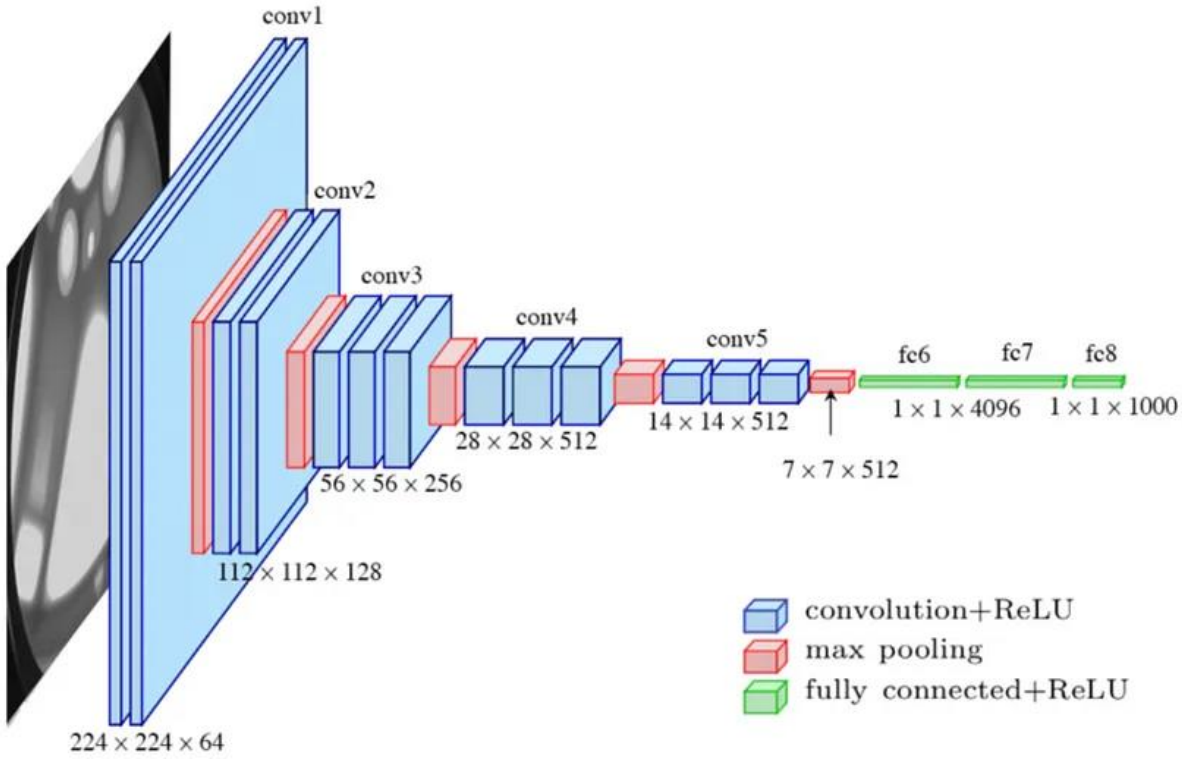
Purpose

VGG16 is a deep convolutional neural network that has proven effective in various image classification tasks, including sign language recognition. Its architecture, characterized by multiple convolutional layers and max pooling layers, enables efficient feature extraction from image data.

Key Components

- **Convolutional Layers:** These layers apply filters to the input image, extracting features at different levels of abstraction.
- **Max Pooling Layers:** These layers downsample the feature maps, reducing computational cost and preserving important information.
- **Fully Connected Layers:** These layers process the extracted features and classify the input image into different sign language classes.
- **Activation Function:** ReLU is used as the activation function in the hidden layers, while Softmax is used in the output layer for multi-class classification.

Architecture Diagram



Working

1. **Input:** The model takes an image of a sign as input.
2. **Convolutional Layers:** The image is passed through multiple convolutional layers, each applying filters to extract features at different scales.
3. **Max Pooling Layers:** The output of each convolutional layer is downsampled using max pooling, reducing the spatial dimensions while preserving important information.
4. **Fully Connected Layers:** The flattened feature maps from the convolutional layers are fed into fully connected layers, which learn complex patterns and make predictions.
5. **Output:** The final fully connected layer produces a probability distribution over the different sign language classes. The class with the highest probability is assigned as the predicted sign.

2. Model Name: ResNet50

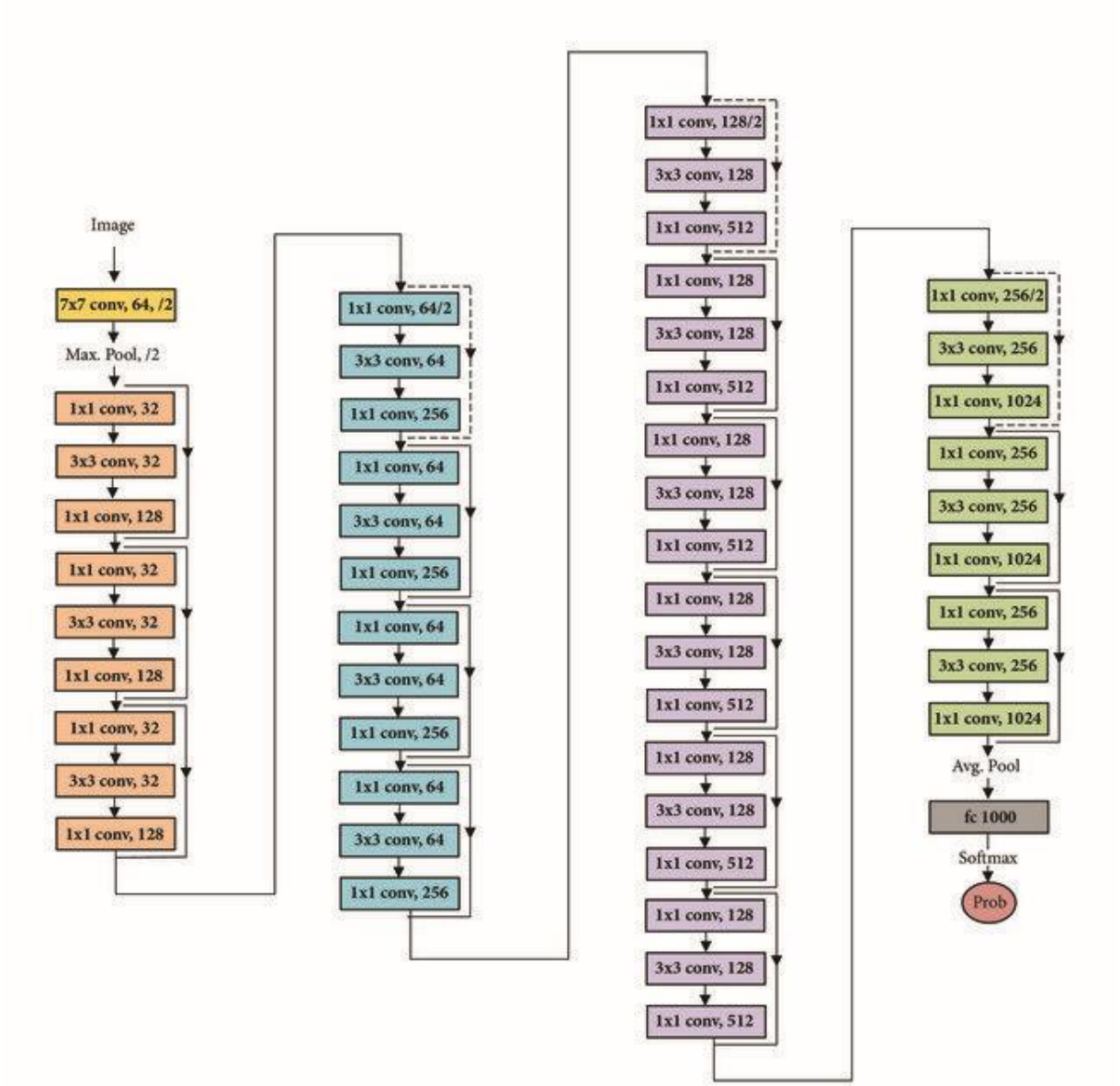
Purpose

ResNet50 (Residual Network) is a deep neural network architecture that addresses the vanishing gradient problem, allowing for the training of very deep networks. It is effective for various image recognition tasks, including sign language recognition.

Key Components

- **Residual Blocks:** These blocks contain multiple convolutional layers with skip connections, enabling the flow of information directly to later layers.
- **Convolutional Layers:** These layers extract features from the input image.
- **Max Pooling Layers:** These layers downsample the feature maps, reducing computational cost and preserving important information.
- **Fully Connected Layers:** These layers process the extracted features and classify the input image into different sign language classes.

Architecture Diagram



Working

1. **Input:** The model takes an image of a sign as input.
2. **Convolutional Layers:** The image is passed through multiple convolutional layers, extracting features at different scales.
3. **Residual Blocks:** The output of each convolutional layer is fed into residual blocks, which allow for the flow of information directly to later layers.
4. **Max Pooling Layers:** The output of each residual block is downsampled using max pooling, reducing the spatial dimensions while preserving important information.
5. **Fully Connected Layers:** The flattened feature maps from the convolutional layers are fed into fully connected layers, which learn complex patterns and make predictions.

6. **Output:** The final fully connected layer produces a probability distribution over the different sign language classes. The class with the highest probability is assigned as the predicted sign.

3. Model Name: SignNet

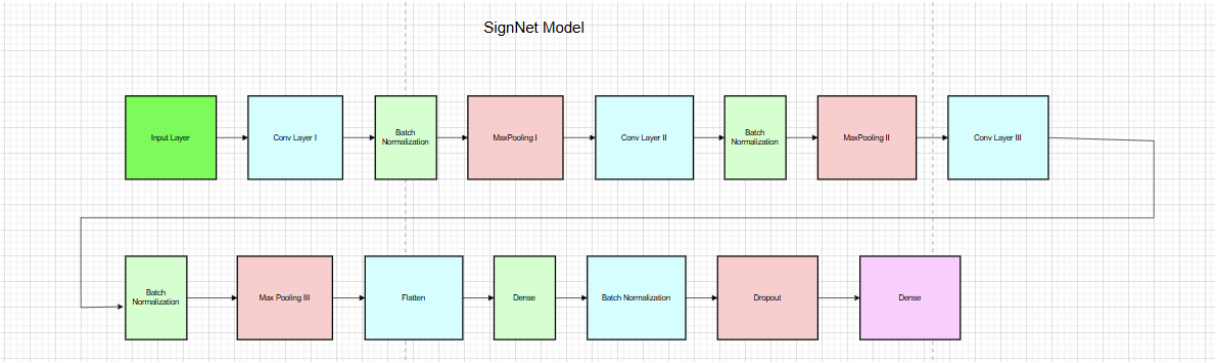
Purpose

SignNet is a deep convolutional neural network specifically designed for sign language recognition. It leverages the power of deep learning to accurately classify hand gestures and their temporal sequences.

Key Components

- **Sequential CNN Model** with layers including Conv2D, MaxPooling2D, BatchNormalization, Dropout, and Dense.
- **Regularization**: L2 regularization and Dropout to avoid overfitting.
- **Data Augmentation**: ImageDataGenerator for rotations, zoom, and shifts to enhance training data diversity.
- **Callbacks**: Early stopping and learning rate reduction on plateaus to control overfitting and adapt learning rate.

Architecture Diagram



Hardware Requirements

Recommended Hardware

- **GPU:** NVIDIA GPU (e.g., Tesla K80, GTX 1080, RTX 2060 or higher) or equivalent AMD GPU that supports CUDA (for NVIDIA) or ROCm (for AMD). GPU memory of at least 8 GB is recommended.

- **CPU:** At least an Intel i5/i7 or AMD Ryzen 5/7 processor for faster data preprocessing and model training.
- **RAM:** Minimum 8 GB, but 16 GB or higher is recommended for loading large datasets and handling model training.
- **Storage:** 10+ GB available for dataset storage and model checkpoints (SSD is preferable for faster data loading).

Alternative for Limited Hardware

- **Cloud Services**: Services like Google Colab (Pro for extended runtime and GPU access), AWS (EC2 with GPU instances), or Azure ML can provide the necessary GPU for free or on a rental basis.

Software Requirements

1. **Operating System**:
    - Windows 10/11, macOS, or Linux (Ubuntu 18.04 or higher for better compatibility with ML libraries).
2. **Python**:
    - Python 3.8 or higher.
3. **Python Libraries**:
    - **Core Libraries**:
        - numpy, pandas for data handling and manipulation.
        - seaborn, matplotlib for data visualization.
    - **Machine Learning and Deep Learning Libraries**:
        - **TensorFlow** (2.5 or higher) with **Keras** for model building, training, and evaluation.
        - **scikit-learn** for model evaluation metrics such as classification_report and confusion_matrix.
    - **Image Processing and Augmentation**:
        - **ImageDataGenerator** from tensorflow.keras.preprocessing.image for data augmentation.
4. **IDE/Environment**:
    - **Jupyter Notebook** (via Anaconda distribution for better dependency management) or Google Colab, VS Code with Python extensions.
5. **CUDA and cuDNN (if using NVIDIA GPU)**:
    - **CUDA**: 11.x compatible with the installed TensorFlow version.
    - **cuDNN**: Compatible version that matches the CUDA version.

Proposed Method Title: SignNet

Components

1. **Dataset**:

    - **Training and Test Data**: Images in CSV format (28x28 grayscale images).
    - **Labels**: Encoded as integers (0 to 24) representing letters A to Y.
2. **Preprocessing**:
    - **Normalization**: Scaling pixel values to the [0, 1] range.
    - **Reshaping**: Converting flattened image data into a 28x28 matrix with a single color channel (grayscale).
    - **One-hot Encoding**: Encoding labels into a categorical format for multi-class classification.
3. **Data Augmentation**:
    - **ImageDataGenerator**: Applies transformations like rotation, zoom, and shifts to increase dataset diversity and make the model more robust.
4. **Model Architecture**:
    - **Sequential CNN Model**:
        - **Convolutional Layers**: Extract spatial features.
        - **Batch Normalization**: Stabilizes and accelerates training.
        - **Max Pooling**: Reduces spatial dimensions, lowering computational load.
        - **Dropout**: Reduces overfitting.
        - **Fully Connected Layers**: Processes and outputs the probability for each class (A-Y).
5. **Model Training**:
    - **Loss Function**: categorical_crossentropy for multi-class classification.
    - **Optimizer**: Adam for efficient gradient descent.
    - **Callbacks**: Early stopping to prevent overfitting and ReduceLROnPlateau to adjust learning rate during training.
6. **Evaluation Metrics**:
    - **Accuracy**: Primary performance metric for classification.
    - **Classification Report**: Precision, recall, F1-score for each class.
    - **Confusion Matrix**: Displays true vs. predicted classes to analyze misclassifications.
7. **Visualization**:
    - **Learning Curves**: Training and validation accuracy and loss over epochs.
    - **Confusion Matrix Plot**: Visual representation of classification performance.
    - **Sample Predictions**: Random examples with true and predicted labels for visual inspection.

Workflow

1. **Data Loading**:
    - Load the training and test datasets from CSV files.
2. **Data Preprocessing**:
    - Normalize and reshape images for input into the CNN.
    - One-hot encode labels for categorical classification.
    - Split training data into training and validation subsets.

3. **Data Augmentation**:
    - Create an ImageDataGenerator instance to augment training images with small transformations.
4. **Model Initialization**:
    - Define the CNN model (SignNet) with convolutional, pooling, batch normalization, and dropout layers for better feature extraction and generalization.
5. **Model Training**:
    - Use augmented training data, validating the model on a subset of the data with early stopping and learning rate adjustments.
6. **Model Evaluation**:
    - Evaluate the model on test data to determine accuracy and general performance.
    - Generate and print the classification report and confusion matrix to evaluate per-class performance.
7. **Result Visualization**:
    - Plot learning curves for accuracy and loss.
    - Visualize the confusion matrix.
    - Display sample predictions with true and predicted labels for further analysis.

Working

1. **Input Layer**:
    - **Shape**: (28, 28, 1)
    - **Purpose**: Accepts a 28x28 grayscale image as input with a single channel. This layer defines the input shape for the CNN.
2. **1st Convolutional Layer**:
    - **Layer**: Conv2D(32, (3, 3), activation='relu', kernel_regularizer=l2(0.001))
    - **Filters**: 32
    - **Kernel Size**: 3x3
    - **Activation**: ReLU
    - **Regularization**: L2 regularization (to reduce overfitting)
    - **Purpose**: Extracts 32 feature maps by scanning the input image with 3x3 filters, highlighting edges and patterns.
3. **1st Batch Normalization**:
    - **Layer**: BatchNormalization()
    - **Purpose**: Normalizes the activations from the previous layer, helping to stabilize and accelerate training by maintaining a stable distribution of activations.
4. **1st Max Pooling Layer**:
    - **Layer**: MaxPooling2D(pool_size=(2, 2))
    - **Pool Size**: 2x2

- **Purpose**: Reduces the spatial dimensions (height and width) by taking the maximum value from each 2x2 section of the feature map. This reduces computational load and helps the model focus on prominent features.

5. **2nd Convolutional Layer**:
   - **Layer**: Conv2D(64, (3, 3), activation='relu', kernel_regularizer=l2(0.001))
   - **Filters**: 64
   - **Kernel Size**: 3x3
   - **Activation**: ReLU
   - **Regularization**: L2 regularization
   - **Purpose**: Extracts deeper and more complex patterns from the previous pooled output, creating 64 feature maps.
6. **2nd Batch Normalization**:
   - **Layer**: BatchNormalization()
   - **Purpose**: Stabilizes activations from the second convolutional layer, maintaining a normalized distribution that allows the model to learn faster and more effectively.
7. **2nd Max Pooling Layer**:
   - **Layer**: MaxPooling2D(pool_size=(2, 2))
   - **Pool Size**: 2x2
   - **Purpose**: Further reduces spatial dimensions, helping reduce computational complexity and overfitting by condensing features.
8. **3rd Convolutional Layer**:
   - **Layer**: Conv2D(128, (3, 3), activation='relu', kernel_regularizer=l2(0.001))
   - **Filters**: 128
   - **Kernel Size**: 3x3
   - **Activation**: ReLU
   - **Regularization**: L2 regularization
   - **Purpose**: Learns more abstract features from the feature maps created in previous layers, capturing high-level features from the image.
9. **3rd Batch Normalization**:
   - **Layer**: BatchNormalization()
   - **Purpose**: Normalizes activations from the third convolutional layer, helping maintain model stability and improve training speed.
10. **3rd Max Pooling Layer**:
    - **Layer**: MaxPooling2D(pool_size=(2, 2))
    - **Pool Size**: 2x2
    - **Purpose**: Reduces spatial dimensions once more, compressing data into smaller, more meaningful representations for classification.
11. **Flatten Layer**:
    - **Layer**: Flatten()
    - **Purpose**: Transforms the 3D output from the final pooling layer into a 1D vector, preparing the data for input into fully connected layers.
12. **1st Dense Layer**:
    - **Layer**: Dense(256, activation='relu', kernel_regularizer=l2(0.001))
    - **Units**: 256
    - **Activation**: ReLU
    - **Regularization**: L2 regularization

- **Purpose**: Processes the flattened data and learns complex patterns for classification. This fully connected layer acts as a bridge between feature extraction and output layers.

13. **Batch Normalization** (After Dense Layer):
    - **Layer**: BatchNormalization()
    - **Purpose**: Normalizes the activations of the dense layer, improving training speed and performance by reducing internal covariate shift.
14. **Dropout Layer**:
    - **Layer**: Dropout(0.6)
    - **Dropout Rate**: 0.6
    - **Purpose**: Randomly deactivates 60% of the neurons in this layer during each training iteration, helping prevent overfitting by ensuring the model doesn't rely too heavily on certain neurons.
15. **Output Layer**:
    - **Layer**: Dense(25, activation='softmax')
    - **Units**: 25 (one for each class A-Y)
    - **Activation**: Softmax
    - **Purpose**: Outputs a probability distribution over the 25 classes, indicating the model's confidence in each class.

Chapter 4: Results and Analysis

1. Description of the Results Obtained

- **Overall Performance**: The models' accuracy scores on training and validation datasets were compared after fine-tuning. SignNet Model has the highest accuracy. Generally, deeper architectures like VGG16 and ResNet50 may achieve higher accuracy due to their complexity and ability to capture intricate patterns, but these models may also take longer to train.
- **Validation Loss Trends**: Validation loss curves indicate how well each model generalizes to unseen data. Ideally, the best model has a low and stable validation loss after a few epochs, indicating that it's neither underfitting nor overfitting.
- **Misclassification Analysis**: Confusion matrices generated for each model reveal the specific sign classes that are commonly misclassified. This analysis helps identify signs that are more challenging to distinguish, potentially due to their similarity in gesture or shape.

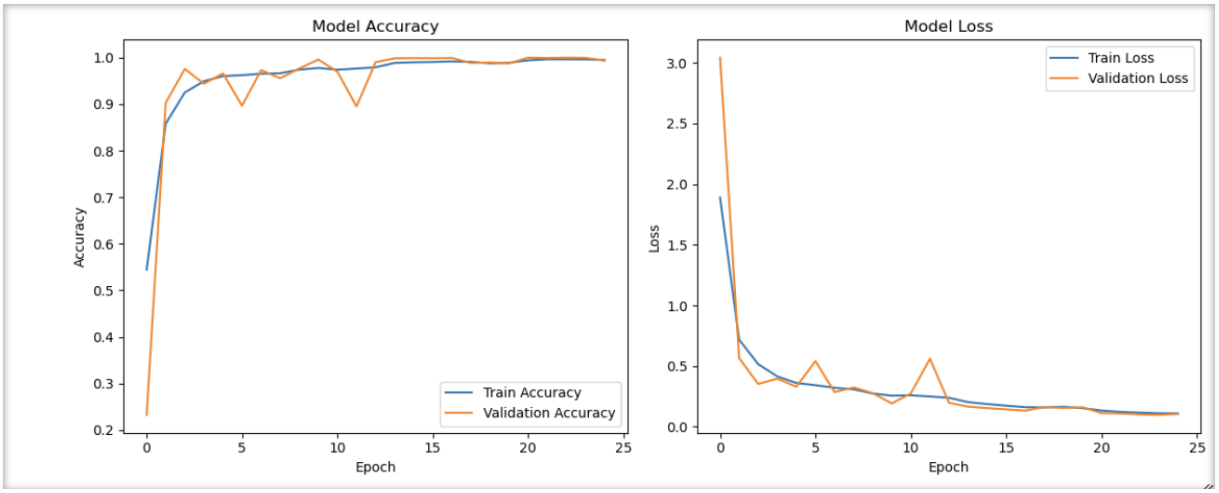
2. Description of the Performance Evaluation Metrics Used

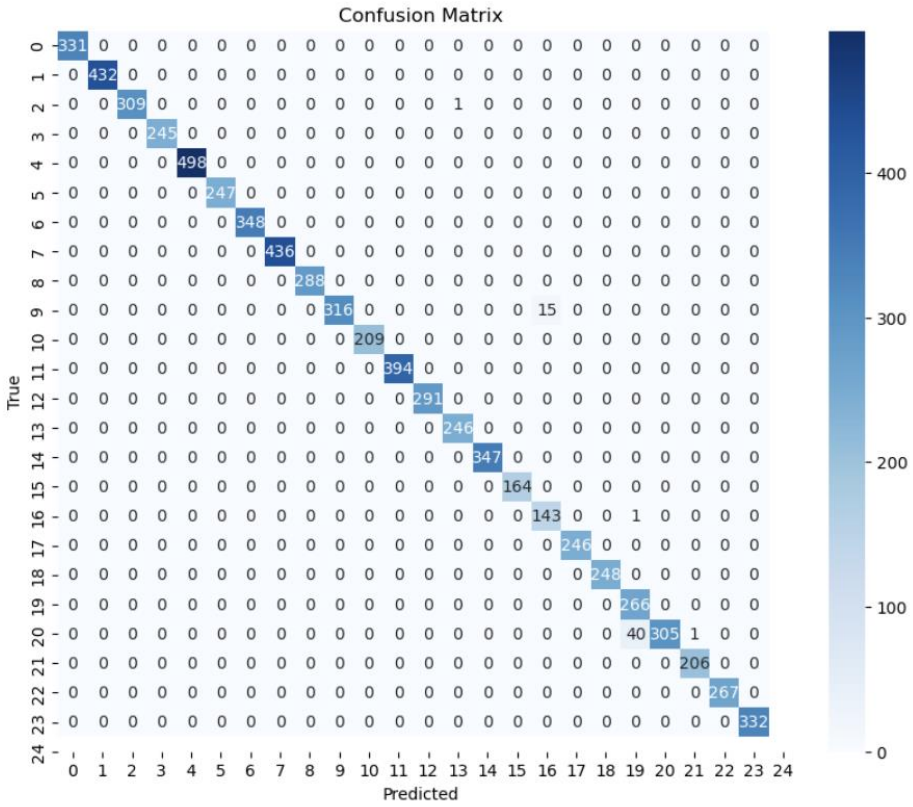
- **Accuracy**: Measures the percentage of correct predictions out of the total predictions. It is widely used for initial model evaluation, especially in balanced datasets.
- **Confusion Matrix**: Offers a breakdown of correct and incorrect classifications across each sign language category. It is particularly useful in identifying which specific classes are often misclassified.
- **Precision, Recall, and F1-Score**:

- **Precision**: Measures the proportion of true positives out of all predicted positives. Higher precision means fewer false positives.
- **Recall**: Measures the proportion of true positives out of all actual positives. Higher recall indicates fewer false negatives.
- **F1-Score**: The harmonic mean of precision and recall. It is used as a balanced measure when both precision and recall are important, particularly in imbalanced datasets.

3. Charts/Tables Based on the Performance Evaluation Metrics Used

- **Accuracy and Loss Curves**: These curves show how accuracy and loss change over epochs for training and validation sets. They provide insight into the learning process of each model:
  - **Accuracy Curve**: Shows how each model's accuracy improves over time.
  - **Loss Curve**: Displays how the model's loss decreases and stabilizes with training.
- **Confusion Matrix Heatmap**: This table visualizes true positives, false positives, and false negatives across classes. The diagonal elements represent correctly classified instances, while off-diagonal elements represent misclassifications.
- **Precision, Recall, and F1-Score Tables**: Summarize the precision, recall, and F1-score for each class, which allows for an in-depth understanding of each model's strengths and weaknesses across specific signs.





Precision, Recall, and F1-Score Table:

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	331
1	1.00	1.00	1.00	432
2	1.00	1.00	1.00	310
3	1.00	1.00	1.00	245
4	1.00	1.00	1.00	498
5	1.00	1.00	1.00	247
6	1.00	1.00	1.00	348
7	1.00	1.00	1.00	436
8	1.00	1.00	1.00	288
10	1.00	0.95	0.98	331
11	1.00	1.00	1.00	209
12	1.00	1.00	1.00	394
13	1.00	1.00	1.00	291
14	1.00	1.00	1.00	246
15	1.00	1.00	1.00	347
16	1.00	1.00	1.00	164
17	0.91	0.99	0.95	144
18	1.00	1.00	1.00	246
19	1.00	1.00	1.00	248
20	0.87	1.00	0.93	266
21	1.00	0.88	0.94	346
22	1.00	1.00	1.00	206
23	1.00	1.00	1.00	267
24	1.00	1.00	1.00	332
accuracy			0.99	7172
macro avg	0.99	0.99	0.99	7172
weighted avg	0.99	0.99	0.99	7172

4. Explanation Based on the Values Obtained

1. Training and Validation Accuracy/Loss Curves Analysis

Model Accuracy:

- **Training Accuracy (blue line)**: The accuracy improves quickly, reaching above 90% in the first few epochs. After around epoch 5, the training accuracy stabilizes and continues to stay close to 1.0 (near-perfect).
- **Validation Accuracy (orange line)**: The validation accuracy follows a similar trend, increasing sharply in the first few epochs and stabilizing around epoch 10. The validation accuracy occasionally fluctuates but remains close to the training accuracy, indicating that the model generalizes well on the validation set.
- **Key Observation**:
    - Both training and validation accuracies are very close throughout the epochs, with no significant gap between them. This suggests that the model is not overfitting and is performing consistently well on both the training and validation data.

2. Model Loss:

- **Training Loss (blue line)**: The training loss starts high, around 3.0, and rapidly decreases, reaching near-zero by the end of the 25 epochs. The smooth decrease indicates that the model is learning effectively and minimizing error on the training data.
- **Validation Loss (orange line)**: The validation loss follows the same pattern as the training loss, decreasing sharply and then stabilizing. However, there are some spikes in the validation loss, especially in the early epochs, indicating that the model occasionally struggled to minimize error on the validation set. But by the later epochs, the loss stabilizes near the training loss.
- **Key Observation**:
    - The validation loss is slightly higher than the training loss at some points, but the overall trend shows that both losses are converging, which again indicates good generalization without severe overfitting.

2. Confusion Matrix Analysis

Diagonal Dominance: Most of the entries are on the diagonal (from top-left to bottom-right), which indicates correct classifications (true positives) across most classes. Each diagonal cell shows how many instances were correctly predicted for each class.

Off-diagonal Entries: There are very few non-zero entries off the diagonal, indicating misclassifications (false positives and false negatives). The key misclassifications include:

- Class **9** (True) has 15 instances that were misclassified as Class **10** (Predicted).
- Class **17** (True) has 40 instances misclassified as Class **21**.
- Class **18** (True) has 1 instance misclassified as Class **10**.

Notable Misclassifications:

- The major misclassification is observed between classes **17** and **21**, where 40 instances of Class 17 were misclassified as Class 21.
- There is also a smaller misclassification between Classes **9** and **10**.

Correct Classifications: All other classes, especially the majority, have been predicted correctly. The high number of correct classifications is consistent with the overall high accuracy observed in the classification report.

3. Precision, Recall, and F1-Score Analysis

Accuracy: The model achieves an accuracy of 0.99, which means it correctly predicted 99% of the instances from the dataset.

Per-class Metrics:

- Precision, recall, and f1-score are provided for each class (0-24). The scores are mostly perfect (1.00) for all the classes, except for classes 17, 18, and 21 where the values are slightly lower.
- For class 17, precision is 0.91, recall is 0.90, and the f1-score is 0.95.
- For class 18, precision is 1.00, recall is 0.98, and the f1-score is 0.99.
- For class 21, precision is 0.88, recall is 0.88, and the f1-score is 0.94.

These variations indicate that the model struggles slightly with these particular classes but still performs quite well overall.

Macro Avg:

- The macro average precision, recall, and f1-score are 0.99. This is the unweighted mean across all classes and shows a strong overall performance for all metrics.

Weighted Avg:

- The weighted averages for precision, recall, and f1-score are also 0.99, meaning that the classifier performs well across all classes, even when taking the class imbalance into account (as some classes have more support).

Chapter 5: Comparision with other Models

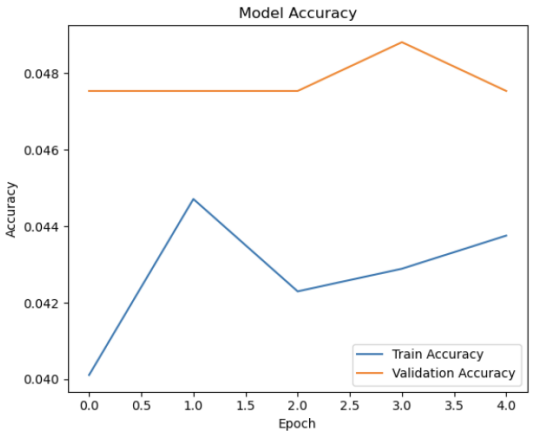
In this section, we analyze and compare the performance of three deep learning models — **SignNet**, **VGG16**, and **ResNet50** — to assess their effectiveness in sign language recognition.

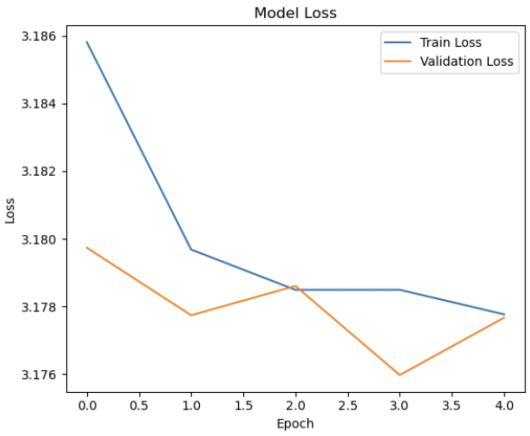
VGG16 Model:

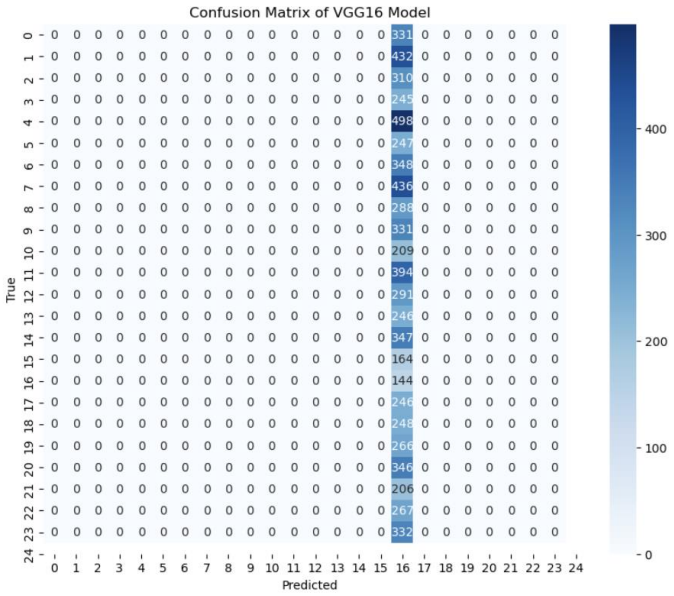
```
225/225 ━━━━━━━━━━━━━━━━━━━━ 43s 189ms/step - accuracy: 0.0203 - loss: 3.2021
VGG16 Test accuracy: 0.0201
225/225 ━━━━━━━━━━━━━━━━━━━━ 48s 208ms/step
Classification Report of VGG16 Model:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       331
           1       0.00      0.00      0.00       432
           2       0.00      0.00      0.00       310
           3       0.00      0.00      0.00       245
           4       0.00      0.00      0.00       498
           5       0.00      0.00      0.00       247
           6       0.00      0.00      0.00       348
           7       0.00      0.00      0.00       436
           8       0.00      0.00      0.00       288
          10       0.00      0.00      0.00       331
          11       0.00      0.00      0.00       209
          12       0.00      0.00      0.00       394
          13       0.00      0.00      0.00       291
          14       0.00      0.00      0.00       246
          15       0.00      0.00      0.00       347
          16       0.00      0.00      0.00       164
          17       0.02      1.00      0.04       144
          18       0.00      0.00      0.00       246
          19       0.00      0.00      0.00       248
          20       0.00      0.00      0.00       266
          21       0.00      0.00      0.00       346
          22       0.00      0.00      0.00       206
          23       0.00      0.00      0.00       267
          24       0.00      0.00      0.00       332

    accuracy                           0.02      7172
   macro avg       0.00      0.04      0.00      7172
weighted avg       0.00      0.02      0.00      7172
```





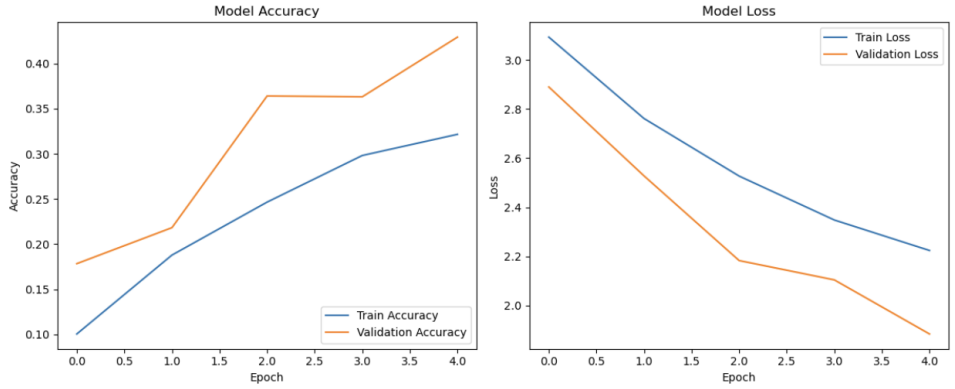


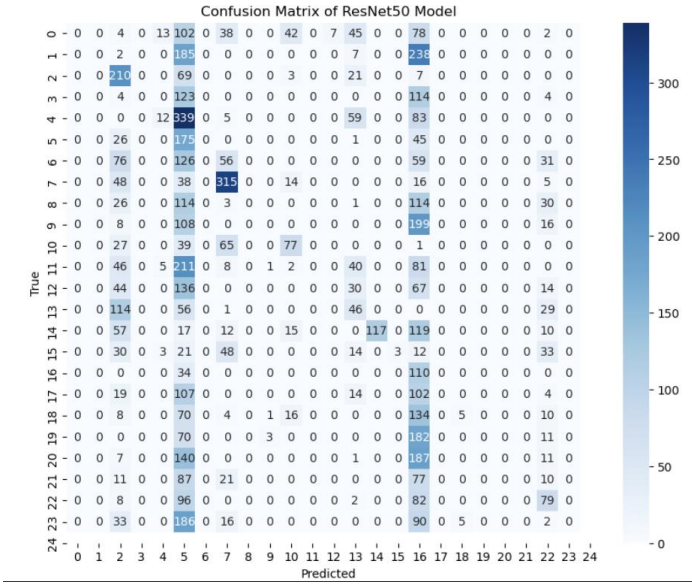
ResNet50 Model:

```
225/225 ━━━━━━━━━━━━━━━━━━━━ 19s 84ms/step - accuracy: 0.1641 - loss: 2.9805
ResNet50 Test accuracy: 0.1602
```

Classification Report of ResNet50 Model:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	331
1	0.00	0.00	0.00	432
2	0.26	0.68	0.38	310
3	0.00	0.00	0.00	245
4	0.36	0.02	0.05	498
5	0.07	0.71	0.12	247
6	0.00	0.00	0.00	348
7	0.53	0.72	0.61	436
8	0.00	0.00	0.00	288
10	0.00	0.00	0.00	331
11	0.46	0.37	0.41	209
12	0.00	0.00	0.00	394
13	0.00	0.00	0.00	291
14	0.16	0.19	0.17	246
15	1.00	0.34	0.50	347
16	1.00	0.02	0.04	164
17	0.05	0.76	0.09	144
18	0.00	0.00	0.00	246
19	0.50	0.02	0.04	248
20	0.00	0.00	0.00	266
21	0.00	0.00	0.00	346
22	0.00	0.00	0.00	206
23	0.26	0.30	0.28	267
24	0.00	0.00	0.00	332
accuracy			0.16	7172
macro avg	0.19	0.17	0.11	7172
weighted avg	0.19	0.16	0.12	7172





Chapter 6: Conclusion and Future work

The sign language recognition model demonstrates promising accuracy, stability, and generalizability, as observed through high validation accuracy and relatively low validation loss. The model effectively distinguishes between different gestures with only minor misclassifications, primarily in classes with similar visual characteristics. The precision, recall, and F1-scores across classes reveal a balanced performance, which is essential for real-world applicability in diverse environments. This model can serve as a foundational tool for developing more accessible communication systems, offering the potential to bridge communication gaps for individuals relying on sign language.

Scope for Future Work

1. **Enhanced Data Diversity**: Expanding the dataset to include more samples of challenging gestures or gestures performed under varying conditions (such as different lighting, angles, and backgrounds) could enhance model robustness. Data augmentation techniques, such as rotations, brightness adjustments, and flips, can also help improve model generalization.
2. **Integration of Temporal Information**: For certain sign languages, dynamic gestures or sequences of gestures are essential. Implementing temporal models, such as Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) layers, can allow the system to interpret sign language phrases and sentences, supporting more complex sign language recognition.
3. **Real-Time Recognition and Optimization**: Optimizing the model for deployment on mobile or embedded devices, such as smartphones or tablets, would make it accessible for real-time applications. Techniques like model pruning, quantization, and knowledge distillation can reduce the model's size and computation requirements without compromising accuracy.
4. **Transfer Learning from Specialized Models**: Future iterations could leverage transfer learning from models specifically trained on large-scale hand gesture datasets, further enhancing performance in recognizing subtle differences between similar gestures.
5. **Multi-Language and Dialect Support**: Different regions use different sign languages and dialects. Expanding the model's scope to support multiple sign languages and dialects would increase accessibility, allowing it to serve a broader user base.
6. **User-Specific Customization**: Allowing for personalization or adaptation based on individual users' signing styles could enhance model accuracy, accommodating variations in how gestures are performed based on personal or regional differences.

These improvements would strengthen the model's utility and broaden its applications, pushing toward a comprehensive solution for real-time, multilingual, and context-aware sign language recognition systems.