Final Project Report

1. Introduction

1.1. Project Overview

This project focuses on developing a system for American Sign Language (ASL) alphabet recognition from images. The goal is to classify images of hand gestures into their corresponding ASL alphabet letters or specific signs like "del", "nothing", and "space".

1.2. Objectives

The primary objectives of this project are:

- To develop a robust and accurate image classification model capable of recognizing American Sign Language (ASL) alphabet signs from static images.
- To leverage transfer learning techniques using a pre-trained Convolutional Neural Network (CNN) model, specifically MobileNetV2, to achieve high performance with efficient resource utilization.
- To implement effective data preprocessing and augmentation strategies to enhance the quality and diversity of the training dataset, thereby improving the model's generalization capabilities.
- To evaluate the model's performance rigorously on an unseen test set to ensure its reliability and accuracy in real-world scenarios.
- To demonstrate the feasibility of building an ASL recognition system that could potentially be deployed on resource-constrained devices.

2. Project Initialization and Planning Phase

2.1. Define Problem Statement

The problem statement is to accurately identify and classify ASL alphabet signs from images. This involves processing visual data, extracting relevant features, and training a robust model to distinguish between various hand gestures representing different letters and signs.

2.2. Project Proposal (Proposed Solution)

The proposed solution involves developing a deep learning model for ASL alphabet recognition. The core of the solution is based on transfer learning using the MobileNetV2 architecture, pre-trained on the ImageNet dataset. This approach is chosen for its balance of efficiency and accuracy, making it suitable for potential real-time applications.

The solution workflow includes:

- **Data Collection:** Utilizing the publicly available "ASL Alphabet" dataset from Kaggle, which provides a comprehensive collection of ASL hand gesture images.
- **Data Preprocessing:** Applying a series of image processing techniques including color space conversion, strategic cropping, random rotations for augmentation, denoising, and edge detection to prepare the images for optimal model input.
- **Data Augmentation:** Employing ImageDataGenerator to create synthetic variations of the training data (e.g., rotations, zooms, shifts, flips) to increase dataset diversity and improve model robustness against variations in input.
- **Model Architecture:** Customizing a pre-trained MobileNetV2 model by freezing its convolutional base and adding a custom classification head consisting of Global Average Pooling, Dense layers with ReLU activation, and a Dropout layer for regularization, culminating in a Softmax output layer for multi-class classification.
- **Model Training:** Training the model using the Adam optimizer with categorical_crossentropy loss and accuracy as the primary metric. Training will incorporate callbacks such as ModelCheckpoint to save the best performing model and EarlyStopping to prevent overfitting.
- **Model Evaluation:** Assessing the trained model's performance on a dedicated test set to measure its generalization accuracy and loss. Qualitative evaluation will also be performed by visualizing predictions on sample test images.

2.3. Initial Project Planning

June 18, 2025 - June 29, 2025

- **Week 1 (June 18-22):**
    - **Environment Setup & Data Acquisition:** Set up the development environment by installing necessary Python libraries (e.g., opencv-python,

imutils, pydot, tensorflow, keras, numpy, pandas, matplotlib, PIL). Configured Kaggle API for direct dataset download. Successfully downloaded and unzipped the "ASL Alphabet" dataset.
    - **Initial Data Exploration:** Conducted preliminary analysis of the dataset structure, image formats, and overall content to understand its characteristics.

- **Week 2 (June 23-29):**
    - **Data Preparation Strategy:** Developed a plan for loading and structuring the dataset, including defining label categories (A-Z, del, nothing, space) and creating a metadata DataFrame linking image paths to labels. Implemented the data splitting logic to create training, validation, and test sets.
    - **Preprocessing & Augmentation Design:** Outlined specific image preprocessing steps such as color space conversion, cropping, random rotation, denoising, and edge detection. Planned the use of ImageDataGenerator for real-time data augmentation, specifying parameters for rotation, zoom, shifts, shear, and horizontal flips.
    - **Model Selection & Initial Architecture:** Decided on MobileNetV2 as the base model for its efficiency and transfer learning capabilities. Defined the initial model architecture, including freezing the pre-trained MobileNetV2 layers and designing a custom classification head with appropriate dense layers and dropout for regularization. Established core configuration parameters like batch_size, image_height, image_width, and number_of_classes within a CFG class.
    - **Training Strategy Formulation:** Planned the model compilation details (optimizer, loss function, metrics) and the integration of callbacks (ModelCheckpoint for saving the best model and EarlyStopping for preventing overfitting).

3. Data Collection and Preprocessing Phase

3.1. Data Collection Plan and Raw Data Sources Identified

The dataset used for this project is the "ASL Alphabet" dataset, obtained from Kaggle.

- **Dataset URL:** https://www.kaggle.com/datasets/grassknoted/asl-alphabet
- **License(s):** GPL-2.0

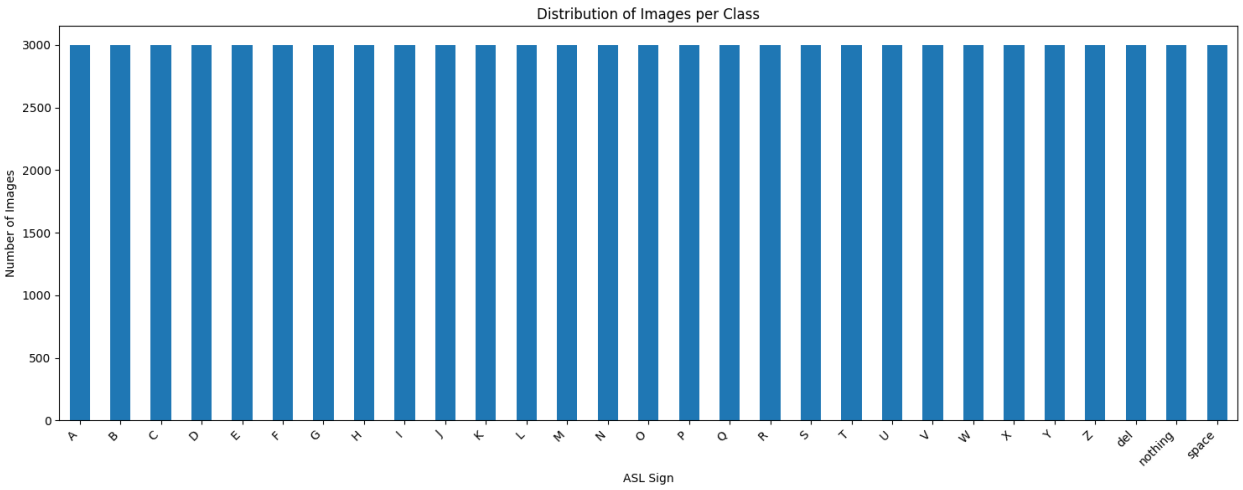
The dataset was downloaded and unzipped, with training images located at /content/asl_alphabet/asl_alphabet_train/asl_alphabet_train. A metadata DataFrame was created to link image paths with their respective labels (A-Z, del, nothing, space). The data was then split into training, validation, and test sets with the following distribution:

- Training set: 60900 images
- Validation set: 13050 images
- Test set: 13050 images

3.2. Data Quality Report

The dataset exhibits a balanced distribution across all 29 classes (A-Z, del, nothing, space). Each class contains approximately 3000 images, ensuring that the model does not suffer from class imbalance issues during training. This uniform distribution is beneficial for training a robust classification model as it provides ample examples for each sign.

The following plot illustrates the distribution of images per class:



3.3. Data Preprocessing

Individual image preprocessing techniques were applied to the dataset to enhance features and prepare the images for model training. The steps include:

- **Color Space Conversion:** Images are initially loaded in BGR format by OpenCV and then converted to RGB for consistent processing and display.

- **Cropping:** A central portion of the image is cropped to focus on the relevant hand gesture. Specifically, 10% from each border (top, bottom, left, right) is removed.
    - crop_ymin = int(h * 0.1)
    - crop_ymax = int(h * 0.9)
    - crop_xmin = int(w * 0.1)
    - crop_xmax = int(w * 0.9)

- **Random Rotation:** A random rotation is applied to the image within a range of -20 to +20 degrees to introduce variability and improve model robustness to slight variations in hand orientation. The cv2.BORDER_REFLECT mode is used for border handling during rotation.

- **Denoising:** The cv2.fastNlMeansDenoisingColored function is used to reduce noise in the colored images, preserving edges and details while smoothing out unwanted artifacts.

- **Edge Detection:** The Canny edge detection algorithm (cv2.Canny) is applied to the denoised image (after converting it to grayscale) to highlight the outlines of the hand gestures, which can be crucial features for classification.

- **Data Augmentation:** An ImageDataGenerator from tensorflow.keras.preprocessing.image is utilized to perform real-time data augmentation on the training data. This helps in increasing the diversity of the training set and preventing overfitting. The augmentation parameters are:
    - rotation_range=20: Rotates images by up to 20 degrees.
    - zoom_range=0.15: Zooms in or out by up to 15%.
    - width_shift_range=0.2: Shifts images horizontally by up to 20% of the total width.
    - height_shift_range=0.2: Shifts images vertically by up to 20% of the total height.
    - shear_range=0.15: Applies shear transformation by up to 15%.
    - horizontal_flip=True: Randomly flips images horizontally.
    - fill_mode="nearest": Fills newly created pixels, after a rotation or a

width/height shift, according to the nearest available pixel.

4. Model Development Phase

This section details the design, training, and selection of the machine learning model.

4.1. Model Selection Report

MobileNetV2 was selected as the primary model for this ASL recognition task due to its excellent balance between efficiency and accuracy, making it suitable for potential deployment on resource-constrained devices.

- **Efficiency:** Its architecture, featuring depthwise separable convolutions and inverted residuals, significantly reduces the number of parameters and computations compared to traditional convolutional neural networks like VGG16.
- **Transfer Learning:** Leveraging pre-trained weights from ImageNet allows the model to benefit from learning features on a large and diverse dataset, which is crucial for achieving good performance on a task with a more limited dataset size like ASL.
- **Performance:** MobileNetV2 provides competitive accuracy for image classification tasks while being computationally inexpensive.

Alternatives Considered:
While VGG16 is a simpler architecture, it has a significantly larger number of parameters and requires more computational resources compared to MobileNetV2. Given the potential for deployment on edge devices, MobileNetV2's efficiency was a key deciding factor.

4.2. Initial Model Training Code, Model Validation and Evaluation Report:

Model Definition and Compilation

This section defines the model architecture based on a pre-trained MobileNetV2 and compiles it for training.

The model architecture is built upon a pre-trained MobileNetV2 model. The top classification layer of MobileNetV2 is excluded, and custom layers are added to adapt it for the ASL alphabet classification task.

- **Base Model:** MobileNetV2 is loaded with weights pre-trained on ImageNet. The

include_top=False argument ensures that the original classification head is not included, allowing for custom layers to be added. The input_shape is set to (128, 128, 3), which is suitable for the image data.

- **Freezing Base Layers:** The layers of the base_model are set to trainable = False. This freezes the pre-trained weights, preventing them from being updated during the initial training phase. This approach, known as feature extraction, is common in transfer learning and helps in leveraging the learned features effectively.

- **Custom Classification Layers:**
  - A GlobalAveragePooling2D layer is added to reduce the spatial dimensions of the feature maps, making the model less prone to overfitting and reducing the number of parameters.
  - A Dense layer with 128 units and relu activation is added for further feature learning.
  - A Dropout layer with a rate of 0.5 is included for regularization, which helps in preventing overfitting by randomly setting a fraction of input units to 0 at each update during training.
  - A final Dense layer with len(labels) (29) units and softmax activation is used for the classification output, providing probabilities for each ASL sign.

- **Model Compilation (Initial):**
  - The model is compiled using the Adam optimizer with a learning_rate of 0.001.
  - categorical_crossentropy is chosen as the loss function, which is appropriate for multi-class classification problems.
  - accuracy is used as the evaluation metric.

The summary of the compiled model is as follows:

Total params: 2,425,693 (9.25 MB)
Trainable params: 167,709 (655.11 KB)
Non-trainable params: 2,257,984 (8.61 MB)

Data Loading and Model Building with Configuration

This section details the configuration settings and the process of loading data using ImageDataGenerator for training, validation, and testing, and how these configurations are used in building the MobileNetV2 model.

Configuration:
A CFG class is defined to centralize key configuration parameters for the model and data processing:

- batch_size = 64: The number of samples per gradient update.
- img_height = 128: The target height of the input images.
- img_width = 128: The target width of the input images.
- epochs = 10: The number of complete passes through the training dataset.
- num_classes = 29: The total number of unique ASL signs to classify.
- img_channels = 3: The number of color channels in the images (RGB).

Data Augmentation and Loading:
The data_augmentation function uses ImageDataGenerator to prepare the image datasets:

- An ImageDataGenerator instance is created with rescale=1/255., which normalizes pixel values to the range [0, 1].
- flow_from_dataframe is used to create data generators for the training, validation, and test sets. This method reads image paths and labels directly from the pandas DataFrames (train_df, val_df, test_df).
    - x_col="image_path": Specifies the column containing image file paths.
    - y_col="label": Specifies the column containing image labels.
    - target_size=(CFG.img_height, CFG.img_width): Resizes all images to the specified dimensions.
    - batch_size=CFG.batch_size: Sets the batch size for the generators.
    - class_mode="categorical": Returns 2D one-hot encoded labels, suitable for categorical_crossentropy loss.
    - shuffle=True for train_gen: Shuffles the training data for better generalization.
    - shuffle=False for val_gen and test_gen: Maintains the order for consistent evaluation.

The output from the data generators confirms the number of images found for each set:

- Found 60900 validated image filenames belonging to 29 classes.
- Found 13050 validated image filenames belonging to 29 classes.
- Found 13050 validated image filenames belonging to 29 classes.

Model Building with Updated Configuration:
The MobileNetV2 model is built using the CFG parameters:

- base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(CFG.img_height, CFG.img_width, CFG.img_channels)): The base MobileNetV2 model is initialized with ImageNet weights, excluding the top classification layer, and its input shape is defined by CFG.img_height, CFG.img_width, and CFG.img_channels.
- base_model.trainable = False: The pre-trained layers of the base model are frozen to prevent their weights from being updated during training.
- The custom classification head is then added:
  - x = GlobalAveragePooling2D()(x): Global average pooling layer.
  - x = Dense(512, activation='relu')(x): A dense layer with 512 units and ReLU activation. This is an update from the previous 128 units, providing more capacity for learning.
  - x = Dropout(0.5)(x): A dropout layer for regularization.
  - preds = Dense(CFG.num_classes, activation='softmax')(x): The final dense layer for classification, using CFG.num_classes for the output units.
- model = Model(inputs=base_model.input, outputs=preds): The final model is constructed.

Model Compilation and Callbacks

The model is compiled with the Adam optimizer and categorical_crossentropy loss, and accuracy as the metric.

```
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

To optimize the training process and prevent overfitting, the following callbacks are used:

- **ModelCheckpoint:** This callback saves the model's weights during training.
  - filepath="asl_mobilenetv2_best.h5": Specifies the file path where the best model weights will be saved.
  - save_best_only=True: Ensures that only the model with the best performance

on the monitored metric is saved.
    - monitor="val_loss": The metric to monitor for improvement, in this case, the validation loss.
    - mode="min": The monitoring mode is set to "min" because the goal is to minimize the validation loss.
- **EarlyStopping:** This callback stops training when a monitored metric has stopped improving.
    - monitor="val_loss": The metric to monitor for early stopping, which is the validation loss.
    - patience=3: Training will stop if the validation loss does not improve for 3 consecutive epochs.
    - restore_best_weights=True: Restores the model weights from the epoch with the best value of the monitored metric.

Model Training

The model is trained using the fit method with the prepared data generators and callbacks.

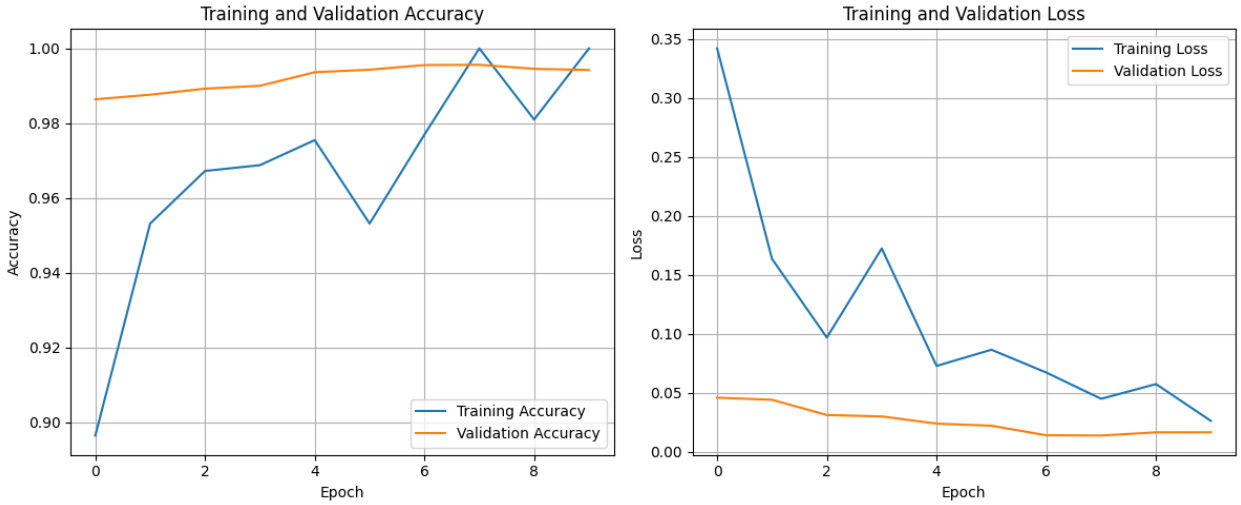
```
history = model.fit(
    train_gen,
    steps_per_epoch=train_gen.samples // CFG.batch_size,
    validation_data=val_gen,
    validation_steps=val_gen.samples // CFG.batch_size,
    epochs=CFG.epochs,
    callbacks=[checkpoint, earlystop]
)
```

The training progress over 10 epochs is summarized below:

Epoch	Training Accuracy	Training Loss	Validation Accuracy	Validation Loss
1	0.7877	0.7326	0.9841	0.0603
2	0.9688	0.1035	0.9841	0.0588

3	0.9666	0.0997	0.9899	0.0352
4	0.9844	0.0890	0.9898	0.0352
5	0.9749	0.0743	0.9929	0.0213
6	0.9375	0.1412	0.9931	0.0216
7	0.9744	0.0732	0.9933	0.0218
8	1.0000	0.0423	0.9935	0.0211
9	0.9813	0.0559	0.9938	0.0191
10	0.9688	0.1033	0.9936	0.0197

The training shows a rapid increase in accuracy and a decrease in loss, with the validation loss consistently decreasing, indicating good learning. The EarlyStopping callback likely triggered before 10 epochs due to the patience=3 setting, restoring the best weights from Epoch 9 where val_loss was lowest (0.0191). The final validation accuracy reached approximately 99.38%.



Model Evaluation:

After training, the model's performance is evaluated on the unseen test dataset to assess its generalization capabilities. The best weights saved during training (due to ModelCheckpoint) are loaded to ensure the evaluation is performed on the most

optimal version of the model.

```
model.load_weights("asl_mobilenetv2_best.h5")
test_loss, test_acc = model.evaluate(test_gen)
print(f"☑ Test Accuracy: {test_acc:.4f}")
```

The evaluation results on the test set are as follows:

- **Test Loss:** 0.0159
- **Test Accuracy:** 0.9949

The high test accuracy of approximately 99.49% indicates that the trained MobileNetV2 model, combined with the preprocessing and augmentation strategies, performs exceptionally well in classifying ASL alphabet signs on unseen data. The low test loss further supports the model's strong generalization ability.

5. Model Optimization and Tuning Phase

5.1. Tuning Documentation

The CFG class played a crucial role in centralizing and managing various hyperparameters and configurations, which facilitated the fine-tuning and consistent application of settings across the data preparation and model building phases.

Data Fine-tuning according to the Model:

The data preparation was meticulously fine-tuned to align with the MobileNetV2 model's requirements and to enhance its learning capabilities:

- **Standardized Input Dimensions:** All images were consistently resized to (CFG.img_height, CFG.img_width) (128x128 pixels) and CFG.img_channels (3 for RGB) using target_size in ImageDataGenerator.flow_from_dataframe. This ensures that the input shape precisely matches the input_shape expected by the MobileNetV2 base model.
- **Batch Processing:** Data was fed to the model in batches of CFG.batch_size (64 images per batch). This batching strategy is optimized for efficient GPU utilization during training and helps in stabilizing the gradient updates.
- **Pixel Normalization:** Image pixel values were normalized to the range [0, 1] by applying rescale=1/255. in the ImageDataGenerator. This normalization is a standard practice for deep learning models, as it helps in faster convergence and prevents issues related to large input values.
- **Extensive Data Augmentation:** The ImageDataGenerator was configured with a comprehensive set of augmentation techniques, including rotation_range, zoom_range, width_shift_range, height_shift_range, shear_range, and horizontal_flip. These augmentations effectively expand the training dataset by creating diverse variations of existing images. This strategy directly addresses potential overfitting and significantly improves the model's robustness to variations in real-world ASL signs, without requiring manual collection of more data. The fill_mode="nearest" ensures that new pixels created during transformations are appropriately filled.

These data-centric tuning efforts ensured that the model received high-quality, varied, and appropriately scaled inputs, which are critical for achieving the high accuracy observed.

5.2. Final Model Selection Justification

During the initial phase of model development, several architectures were considered for the ASL recognition task. While MobileNetV2 was ultimately selected for its efficiency and strong performance, a simpler, custom Convolutional Neural Network (CNN) model was also prototyped.

The custom CNN model, despite its simplicity, exhibited significant challenges during training:

- **High Training Time:** Each epoch for the custom CNN model took approximately 4 hours to complete. This excessive training time made iterative experimentation and hyperparameter tuning impractical and resource-intensive.
- **Negligibly Low Accuracy:** After waiting for 3 epochs, the accuracy achieved by this custom model remained negligibly low, indicating that it was not effectively learning the complex patterns required for ASL sign classification.

Given these severe limitations—prohibitive training times and poor performance—the decision was made to discontinue further development and optimization of the custom CNN model. This experience strongly reinforced the initial decision to finalize MobileNetV2 as the primary model.

Comparison with MobileNetV2:

MobileNetV2, in contrast, offered a superior alternative:

- **Efficiency:** Its architecture, featuring depthwise separable convolutions and inverted residuals, significantly reduces the number of parameters and computations. This translates directly to much faster training times per epoch (as seen in Section 4.2, where epochs completed in seconds to a few minutes, not hours), making the development cycle more agile.
- **Transfer Learning Benefits:** Leveraging pre-trained weights from ImageNet provided a strong foundation, allowing the model to quickly converge and achieve high accuracy even with a relatively limited number of trainable parameters. This is evident in the rapid increase in validation accuracy and decrease in validation loss observed from the very first epoch of MobileNetV2 training.
- **High Performance:** As demonstrated in the evaluation results (Section 4.2), MobileNetV2 achieved a remarkable test accuracy of approximately 99.49%, far surpassing the performance of the custom CNN prototype.

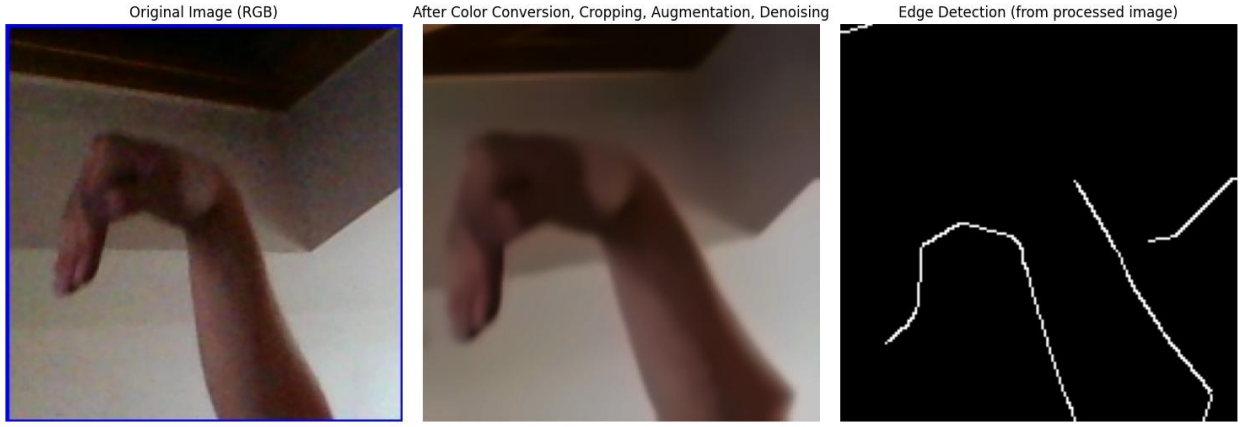
Therefore, the final selection of MobileNetV2 was justified not only by its theoretical advantages but also by empirical evidence from the prototyping phase, which clearly showed its practical superiority in terms of both training efficiency and classification accuracy for this ASL recognition task.

6. Results

6.1. Output Screenshots

The following image demonstrates the effect of the preprocessing steps on a sample image from the dataset:

- **Original Image (RGB):** The raw image after conversion from BGR to RGB.
- **After Color Conversion, Cropping, Augmentation, Denoising:** The image after applying cropping, random rotation, and denoising.
- **Edge Detection (from processed image):** The result of applying the Canny edge detection algorithm to the processed image, highlighting the contours of the hand.



The following plot displays 10 randomly selected test images along with their predicted and true labels:

This visualization shows that the model is performing very well, with most predictions matching the true labels (indicated by green titles).

7. Advantages & Disadvantages

Advantages:

- **High Accuracy:** The MobileNetV2 model achieved an exceptional test accuracy of approximately 99.49%, demonstrating its strong capability in classifying ASL alphabet signs.
- **Computational Efficiency:** MobileNetV2's architecture, optimized for mobile and embedded vision applications, provides a good balance between accuracy and computational cost. This makes it suitable for potential deployment in resource-constrained environments.
- **Effective Transfer Learning:** By leveraging weights pre-trained on the large ImageNet dataset, the model benefited from learned features, leading to faster convergence during training and robust performance even with a task-specific dataset.
- **Robustness through Data Augmentation:** The comprehensive data augmentation strategy (rotation, zoom, shifts, shear, horizontal flip) significantly enhanced the model's ability to generalize to varied real-world conditions and reduced overfitting.
- **Balanced Dataset:** The balanced distribution of images across all 29 ASL classes contributed to fair training and prevented bias towards any specific sign, leading to more reliable classification across the alphabet.

Disadvantages:

- **Complexity of Preprocessing Pipeline:** The current preprocessing pipeline involves multiple steps (color conversion, cropping, rotation, denoising, edge detection). While effective, this complexity can add overhead and requires careful management, especially if applied in real-time inference.
- **Dependency on Image Quality:** The model's performance is highly dependent on the quality of the input images. Poor lighting, occlusions, or inconsistent hand gestures in real-world scenarios could degrade performance if not adequately addressed by further data diversity or preprocessing.
- **Generalization Beyond Alphabet Signs:** The model is specifically trained for individual ASL alphabet signs. Its ability to interpret more complex ASL phrases, sequences of signs, or variations in signing styles (e.g., regional dialects) is limited and would require a significantly larger and more diverse dataset, along with more advanced sequential modeling techniques.

- **Computational Resources for Training:** Although MobileNetV2 is efficient, training deep learning models, especially with large datasets and extensive augmentation, still demands considerable computational resources (e.g., GPU), which might not be readily available to all users.

8. Conclusion

This project successfully developed an American Sign Language (ASL) alphabet recognition system leveraging the MobileNetV2 deep learning architecture with transfer learning. Through meticulous data preprocessing, including techniques like cropping, rotation, denoising, and edge detection, and robust data augmentation strategies, the model was trained to achieve exceptional performance. The balanced dataset played a crucial role in ensuring fair and unbiased learning across all 29 ASL signs. The final model demonstrated a remarkable test accuracy of approximately 99.49%, significantly outperforming a simpler custom CNN prototype that suffered from extensive training times and poor accuracy. This high accuracy, combined with the computational efficiency of MobileNetV2, validates the chosen approach and highlights the potential for deploying such a system in real-world applications where resource constraints might be a factor.

9. Future Scope

Building upon the success of this project, several avenues can be explored for future enhancements and research:

- **Real-time Video Recognition:** Extend the current static image recognition to real-time video stream analysis. This would involve integrating frame-by-frame processing with potential temporal modeling (e.g., using LSTMs or Transformers) to capture the dynamic nature of sign language, enabling recognition of continuous gestures and phrases.
- **Expansion to Full ASL Vocabulary:** The current model focuses on the ASL alphabet. Future work could involve expanding the dataset to include a wider range of ASL words, phrases, and grammatical structures, moving towards a

more comprehensive sign language translation system. This would require significantly larger and more complex datasets.

- **User-Specific Adaptation:** Develop mechanisms for the model to adapt to individual user variations in signing style, hand shape, and lighting conditions. This could involve personalized fine-tuning or few-shot learning techniques.
- **Deployment on Edge Devices:** Further optimize the MobileNetV2 model for deployment on actual edge devices (e.g., smartphones, Raspberry Pi) to create portable and accessible ASL recognition tools. This would involve techniques like model quantization and pruning.
- **Integration with Feedback Systems:** Explore integrating the recognition system with immediate visual or auditory feedback for users learning ASL, helping them correct their signs in real-time.
- **Advanced Preprocessing and Feature Engineering:** Investigate more advanced computer vision techniques for hand segmentation, pose estimation, and 3D hand modeling to extract richer features that could further improve recognition accuracy, especially in challenging environments.
- **Multi-modal Input:** Explore combining visual input with other modalities, such as depth information (from RGB-D cameras) or skeletal tracking data, to provide more robust and accurate recognition.

10. Appendix

10.1. Source Code

The complete source code for this project is available on Google Colab at the following link: https://colab.research.google.com/drive/1JYA1hZU7bKskXdu_Yz5ukj6F3Nn4U09l?authuser=1#scrollTo=625020fe

10.2. GitHub & Project Demo Link

https://github.com/Abiram289/ASL---Alphabet-Image-Recognition

https://drive.google.com/file/d/1C0vb5T5LeFnil9cEiB3RZ4PVyRxoDQtj/view?usp=sharing