**Sign Language Recognition**

Team Number - 16

**Aswin Raj E K- CB.EN.U4CSE21607**

**Kannan R V - CB.EN.U4CSE21629**

# Mobile Application Progress:

# WhatsApp Image 2024-10-24 at 00.07.55_dae732faWhatsApp Image 2024-10-24 at 00.08.17_8f7265c1WhatsApp Image 2024-10-24 at 00.08.40_96859174WhatsApp Image 2024-10-24 at 00.09.02_714bf9b7

# Problem Statement: The task is to design and implement a deep learning model using CNNs that can accurately recognize and classify hand signs from images. This model will be trained on a dataset of hand sign images and integrated into an Android application developed in Android Studio. The application should be capable of capturing images or video streams via the device camera, processing them through the CNN model, and providing real-time or near real-time predictions. The system will include data preprocessing and augmentation techniques to improve model robustness and will be evaluated based on accuracy, precision, recall, and overall performance across different hand sign classes.

# **Technology/Work:**

# **Technology/Work**: TensorFlow & Keras-based MobileNetV2 for Hand Sign Recognition

# **Description**: A transfer learning-based approach utilizing MobileNetV2, a pre-trained model, fine-tuned for recognizing hand signs. Custom layers are added on top of MobileNetV2, including pooling, dropout, and dense layers for classification. The model is trained using augmented hand sign images, optimizing accuracy and performance.

# **Supported Platforms**: Android, iOS (via TensorFlow Lite), Python environment (for development and training)

# ---

# **Technology/Work**: Image Data Processing & Augmentation

# **Description**: Techniques such as rescaling, rotation, shifting, zooming, and flipping are applied to training images using TensorFlow's `ImageDataGenerator`. This helps improve model generalization and robustness against overfitting, allowing the model to learn from varied representations of hand signs.

# **Supported Platform**s: Python environment

# ---

# **Technology/Work**: Transfer Learning with MobileNetV2

# **Description**: The MobileNetV2 model pre-trained on ImageNet is loaded and used as the base. Transfer learning is employed by freezing the base model layers to retain the pre-trained weights while adding custom layers on top for hand sign classification.

# Supported Platforms: Python environment, Android, iOS (via TensorFlow Lite)

# ---

# **Technology/Work**: GPU Acceleration

# **Description**: The training process leverages GPU acceleration for faster computation, enabling quicker processing of large datasets, training of deep neural networks, and efficient backpropagation.

# Supported Platforms: Python environment with TensorFlow GPU support

# ---

# **Technology/Work**: Model Evaluation & Callbacks

# **Description**: Early stopping, learning rate adjustment, and model checkpointing callbacks are used to enhance model training. These callbacks prevent overfitting, adjust the learning rate based on validation loss, and save the best performing model during training.

# **Supported Platforms**: Python environment

# ---

# **Technology/Work**: Model Saving & Deployment

# **Description**: The trained MobileNetV2 model is saved in `.keras` format, ready for deployment in mobile platforms using TensorFlow Lite. This enables real-time hand sign recognition in Android and iOS applications.

# **Supported Platforms**: Android, iOS (via TensorFlow Lite), Python environment

# ---

# **Technology/Work**: Model Architecture Visualization

# **Description**: The architecture of the MobileNetV2-based model is visualized to provide a detailed overview of the layers and their connectivity. This helps in understanding the flow of data through the layers and debugging.

# **Supported Platforms**: Python environment

# ---

# This detailed breakdown aligns with the components of the code and highlights how each technology or feature is implemented within the TensorFlow and Keras ecosystem for hand sign recognition.

# Application Architecture:

# Hand Sign Recognition Project Workflow Using CNN:

**Import Necessary Libraries**

* 1. **Description**: Import TensorFlow, Keras, and additional libraries for building a custom Convolutional Neural Network (CNN), along with utilities such as ImageDataGenerator for preprocessing and augmentation.
  2. **Tools Used**: TensorFlow, Keras (Conv2D, MaxPooling2D, Flatten, Dense, ImageDataGenerator)

**Load and Preprocess the Dataset**

* 1. **Description**: Load the hand sign dataset from specified directories. Apply preprocessing steps, including resizing, rescaling (to normalize pixel values), and data augmentation techniques like rotation, zooming, shearing, and flipping to make the model more robust.
  2. **Tools Used**: ImageDataGenerator (for augmentation, rescaling)

**Split the Data into Training and Validation Sets**

* 1. **Description**: Automatically split the dataset into training (80%) and validation (20%) sets using ImageDataGenerator with flow\_from\_directory. This ensures the training and validation sets are segregated for the model.
  2. **Tools Used**: ImageDataGenerator (with validation split)

**Build the CNN Architecture**

* 1. **Description**: Construct a custom CNN model consisting of several convolutional and pooling layers to capture spatial features from images. The model includes:
     1. Multiple Conv2D layers for feature extraction.
     2. MaxPooling2D layers to reduce spatial dimensions.
     3. Flatten layer to convert the 2D feature maps to 1D.
     4. Dense layers for classification.
  2. **Tools Used**: Conv2D, MaxPooling2D, Flatten, Dense

**Example CNN Architecture**:

* 1. **Conv2D Layer 1**: 32 filters, 3x3 kernel, ReLU activation.
  2. **MaxPooling2D Layer 1**: Pool size 2x2.
  3. **Conv2D Layer 2**: 64 filters, 3x3 kernel, ReLU activation.
  4. **MaxPooling2D Layer 2**: Pool size 2x2.
  5. **Flatten Layer**: Converts the 2D feature maps into a 1D vector.
  6. **Dense Layer 1**: 128 units, ReLU activation.
  7. **Dropout Layer**: Dropout rate 0.5 to reduce overfitting.
  8. **Output Layer**: Softmax activation for classification into the hand sign categories.

**Compile the Model**

* 1. **Description**: Compile the CNN model with the Adam optimizer, categorical crossentropy as the loss function, and accuracy as the metric. This step sets up the model for training.
  2. **Tools Used**: Adam optimizer, categorical\_crossentropy loss

**Train the Model**

* 1. **Description**: Train the CNN model on the training data while validating on the validation set. Use callbacks like early stopping to prevent overfitting and save the best model. Train for 50 epochs.
  2. **Tools Used**: TensorFlow callbacks (EarlyStopping, ModelCheckpoint, ReduceLROnPlateau), Keras fit function

**Evaluate the Model**

* 1. **Description**: After training, evaluate the CNN model on the validation set to calculate accuracy, loss, and other performance metrics. This gives insight into the model's performance on unseen data.
  2. **Tools Used**: TensorFlow evaluate function

**Save the Model**

* 1. **Description**: Save the trained CNN model in a suitable format for future use, such as a .keras file or as a .tflite file for mobile deployment.
  2. **Tools Used**: model.save (Keras model saving utility)

**Convert the Model to TensorFlow Lite (Optional)**

* 1. **Description**: Convert the CNN model to TensorFlow Lite format for deployment in mobile applications. This is an optional step but is useful if the model is to be used in Android apps.
  2. **Tools Used**: TensorFlow Lite Converter (TFLiteConverter)

**Visualize Model Architecture**

* 1. **Description**: Generate a diagram of the CNN's architecture, showing all layers and their dimensions to better understand how the model processes input.
  2. **Tools Used**: plot\_model (Keras visualization utility)

This CNN-based workflow is designed for hand sign recognition by extracting spatial features from images using custom convolutional layers, training the model, and saving it for deployment.

# IMG_256

# Results:

# ****Training History Plots (Accuracy and Loss)****:

# MobileNetV2:

# 

# 

# Vision Transformer:

# WhatsApp Image 2024-10-24 at 11.03.30_b3e55635

# CNN:

# IMG_256IMG_256

# 2) Class Distribution :

# 

# **Classification Report :**

# **CNN:**

# 

# 

# Vision Transformer:

# WhatsApp Image 2024-10-24 at 11.14.30_7c1e1b8d

# WhatsApp Image 2024-10-24 at 11.11.32_43289e71

# 

# MobileNetV2:

# WhatsApp Image 2024-10-24 at 11.14.11_ddef6ec8

# WhatsApp Image 2024-10-24 at 11.12.16_6e453817

# Model Comparison:

|  |  |
| --- | --- |
| Architecture | Model Accuracy |
| CNN | 97.35 |
| Vision Transformer | 78.00 |
| PDDNet | 66.00 |
| MobileNetV2 | 67.78 |