**RELATED WORK**

**1. Model Choice: a) MobileNetV2**

**MobileNetV2** is a lightweight, efficient convolutional neural network architecture that is widely used for image classification, particularly on resource-constrained devices. It’s based on depthwise separable convolutions, which reduce the number of parameters and computation required, making it faster and less memory-intensive. Here’s a summary of key features:

* **Depthwise Separable Convolutions**: Split each filter operation into separate spatial and depthwise convolutions, significantly reducing computational complexity.
* **Inverted Residuals and Linear Bottlenecks**: These structures make the model more memory-efficient by keeping the activation size small.
* **Pretrained Weights on ImageNet**: By leveraging pretrained weights, the model can benefit from previously learned features, which speeds up training and can improve accuracy for smaller datasets.

**B)CNN**

A Convolutional Neural Network (CNN) is a type of deep learning model specifically designed for analyzing visual data, like images and videos. CNNs are widely used in image classification, object detection, and various other computer vision tasks because they can automatically learn spatial hierarchies of features from the input images. CNNs leverage convolutional layers to detect patterns, such as edges, textures, and shapes, making them highly effective in extracting relevant features. Here’s a summary of key features:

* **Automatic Feature Extraction**: CNNs automatically learn and extract features from images through convolutional layers, reducing the need for manual feature engineering.
* **Spatial Hierarchy of Features**: They capture spatial relationships by using multiple convolutional layers that detect increasingly complex features as data passes through the network, enabling CNNs to recognize patterns at various levels (e.g., edges, shapes, and complex objects).
* **Translation Invariance**: Through pooling layers, CNNs achieve translation invariance, meaning they can detect objects in images regardless of their location, which improves robustness for image recognition tasks.

**2. Data Preprocessing**

Data preprocessing is crucial for ensuring consistency in model inputs and improving model performance. Here's an overview of each step:

* **Loading Images**: This function traverses directories to load image files, organize them by class, and handle any potential loading errors.
* **Removing Null and Outlier Classes**: Classes with insufficient samples (fewer than a defined threshold) are removed to maintain data balance. Balancing class data can prevent model bias toward classes with more samples.
* **Resizing and Normalizing Images**: Resizing all images to 128x128 provides a consistent input shape for the model, while normalizing pixel values to the range [0, 1] scales the input data. Normalization is especially important for models like MobileNetV2 that expect inputs in this range due to pretrained weights.
* **Encoding Labels**: Labels are converted to numerical values with LabelEncoder, a common practice in classification tasks that helps simplify model training. Additionally, encoded labels are stored in a way that is memory-efficient.

**3. Data Augmentation**

Data augmentation techniques were applied to artificially expand the training dataset by creating modified versions of the original images. This approach can enhance model generalization and reduce overfitting:

* **Random Rotations, Width, and Height Shifts**: Minor adjustments to image orientation and position help the model recognize objects under various spatial transformations.
* **Horizontal Flipping and Zooming**: Flipping and zooming in on images simulate real-world variations in object appearance, further aiding generalization.

These transformations are applied using ImageDataGenerator on the fly during model training, thus increasing data diversity without requiring additional storage.

**4. Evaluation and Data Visualization**

Visualization techniques provide insight into model performance:

* **Accuracy Plot**: This plot shows how accuracy changes over epochs for both training and validation sets, helping identify convergence trends or overfitting.

.