**RELATED WORK**

**1. Model Choice: 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.

**2. VGG19:** The VGG19 model is a convolutional neural network (CNN) architecture that has been widely used for image classification and feature extraction tasks. It was introduced by the Visual Geometry Group (VGG) in their 2014 paper titled *"Very Deep Convolutional Networks for Large-Scale Image Recognition."* Here are the **key features of VGG19**:

**Architecture Depth**:

* Contains **19 layers** (16 convolutional + 3 fully connected).
* A deeper architecture compared to earlier models like AlexNet.

**Fixed Kernel Size**:

* Uses **3x3 convolutional kernels** throughout the network.
* Helps in capturing fine features.

**Max Pooling**:

* **2x2 pooling** reduces spatial dimensions after every 2-3 convolutional layers.
* Balances feature extraction and computational efficiency.

**Uniform Design**:

* Consistent use of ReLU activation and doubling filters across layers: **64 → 128 → 256 → 512 → 512**.

**Fully Connected Layers**:

* Three dense layers, with two having **4096 neurons** each and one with **1000 neurons** (for ImageNet classes).

**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.
* **Confusion Matrix**: The confusion matrix reveals how well the model distinguishes between classes by displaying true positives, false positives, and false negatives for each class. This helps identify which classes the model confuses most often, guiding future adjustments or refinements.

**5.Deployment:**

* Frontend : HTML ,CSS
* Backend : Flask