Task 2 : CNN

Image classification is a key task in computer vision, but raw images often need preparation before training a model. Differences in size, lighting, and angles can confuse the model and hurt its performance. In this project, you'll use the Flowers Multiclass Dataset from Kaggle to classify flower species. Preprocessing ensures all images are in a consistent format, which helps the model learn effectively. Data augmentation creates more varied training examples, making the model better at handling new images. You'll build a VGG-style CNN from scratch and fine-tune a pretrained ResNet, comparing how these approaches affect accuracy and reliability.

1. Dataset Loading:

Get the Flowers Multiclass Dataset from Kaggle and load it into your coding environment.

- Step 1: Sign up for a Kaggle account if you don't have one. Go to the Flowers Recognition dataset page and click "Download," or use the Kaggle Hub: :

```
import kagglehub
path=kagglehub.dataset_download("alsaniipe/flowers-multiclass-datasets")
print("Path to dataset files:", path)
```

- Step 2: Unzip the downloaded file and load it into Python using PyTorch or TensorFlow.
- Step 3: If the dataset isn't split, divide it into training (70%), validation (10%), and test (20%) sets.

2. Image Preprocessing:

Prepare the images and labels so they're ready for training.

- Resize: Make all images 224x224 pixels (a common size for CNNs like VGG or ResNet).
- Normalize: Scale pixel values to [0,1] by dividing by 255.
- Labels: Choose between two label formats and experiment to see which works better:
    - One-hot encoding: Convert labels to one-hot vectors (e.g., class 2 of 5 becomes [0, 0, 1, 0, 0]).
    - Label encoding: Use class indices directly (e.g., 0, 1, 2 for each class).
      Note: When choosing a label format, ensure your loss function matches.

3. Data Augmentation:

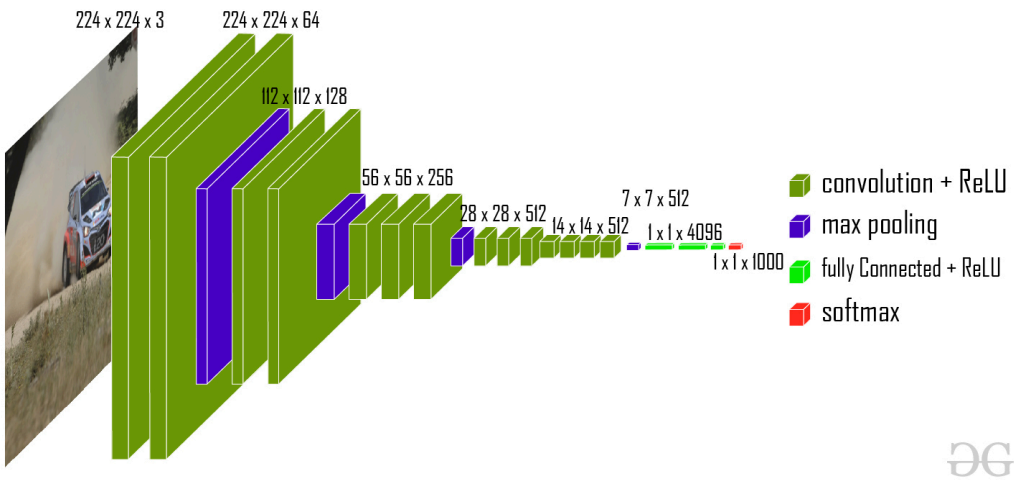
Add variety to the training set only (not validation) to help the model generalize.

- Suggested transformations:
  - Random rotations between -20° and 20°.
  - Horizontal flips (50% chance).
  - Random crops or shifts (e.g., 10% of image size).
  - Slight brightness or contrast changes.
  - other transformations that you found suitable.

4. Building a VGG CNN from Scratch:

Create and train a VGG-style CNN.

- VGG Basics: VGG is a deep network with stacked convolutional layers (e.g., 3x3 filters), max-pooling layers, and fully connected layers at the end. bellow is the VGG-16 architecture also you can use this helper :



- Define the model.
- Train it on the preprocessed and augmented training data for appropriate epochs.
- Evaluate it on the validation set.

5. Fine-Tuning a Pretrained ResNet:

Use a pretrained ResNet (e.g., ResNet50) and fine-tune it in stages.

- Setup: Load ResNet50.

- Freeze Base, Train Head: Freeze all convolutional layers and replace the final layer(s) with a new classifier for flower classes. Train for 5-10 epochs.
- Unfreeze Last Conv Layer: Unfreeze the last convolutional block (e.g., layer4 in PyTorch ResNet50) and train it with the classifier for 5-10 epochs.
- Unfreeze All: Unfreeze all layers and train the whole network for 5-10 epochs.
- Evaluate each method on the validation set.

6. Result Comparison:

Compare the VGG and ResNet models fairly.

- Calculate metrics: accuracy, precision, recall, and F1-score.
- Plot confusion matrices and ROC (Receiver Operating Characteristic) curves using matplotlib or seaborn for deeper insights. Calculate and report the AUC (Area Under the Curve) score for each model. (The ROC curve illustrates the diagnostic ability of a binary classifier by plotting the true positive rate against the false positive rate at various threshold settings, while AUC quantifies the overall performance of the classifier as a single value between 0 and 1, where a higher value indicates better discrimination ability.)

Additional Notes:

- Pick PyTorch or TensorFlow—both work well here.
- Training ResNet fully can take time; use a GPU if possible (e.g., Google Colab's free GPU).
- Play with settings like learning rate (start with 0.001), batch size (e.g., 32), or optimizers (e.g., Adam).
- Track training with loss/accuracy plots (matplotlib) and stop early if the model overfits (e.g., use EarlyStopping in TensorFlow).