📘 ASL Alphabet Image Recognition – Code Demonstration

🧠 Project Summary

This project classifies American Sign Language (ASL) letters (A–Z plus `del`, `space`, and `nothing`) using image data. A pre-trained **VGG16 model** is used for feature extraction, and custom dense layers are added to specialize the network for ASL classification.

🔁 Step-by-Step Code Explanation

✅ 1. Setup & Environment Configuration

- **Kaggle Authentication**: Copy the `kaggle.json` API key to `~/.kaggle/`.
- **Download Dataset**: Use `kaggle datasets download -d grassknoted/asl-alphabet`.
- **Unzip** the contents into a directory called `asl-alphabet`.

📦 2. Dependencies

- TensorFlow/Keras (model building)
- OpenCV & PIL (image loading/processing)
- NumPy, Pandas (data handling)
- Plotly, Matplotlib (visualization)
- `imutils`, `sklearn`, `tqdm`, `glob`

⚙️ 3. Configuration & Seeding

- Hyperparameters are stored in a class `CFG`.

- batch_size = 64
- img_height = 64
- img_width = 64
- epochs = 10
- num_classes = 29

📁 4. Label Creation & Metadata Building

- Uses `string.ascii_uppercase + ['del', 'nothing', 'space']` to define 29 class labels.
- Each label's directory is scanned for image paths and stored in a metadata DataFrame.

✂️ 5. Dataset Splitting

- Uses `train_test_split` (stratified) to split metadata into:
    - 70% Training
    - 15% Validation
    - 15% Testing

🔁 6. Image Data Generators

- Uses `ImageDataGenerator(rescale=1./255)` for normalization.
- Image generators load and batch the image files from disk efficiently.

🧠 7. Model Creation – VGG16 Transfer Learning

- **Base Model**: VGG16 (excluding top classifier).
- **Custom Top Layers**:
    - `Flatten`
    - `Dense(256, relu)` → `Dropout`
    - `Dense(512, relu)` → `Dropout`
    - `Dense(29, softmax)`
- The base VGG16 layers are frozen to retain learned features.

🏋️ 8. Model Compilation & Training

- **Loss Function**: Categorical Crossentropy
- **Optimizer**: Adam
- **Metrics**: Accuracy
- Uses `ModelCheckpoint` to save the best model based on validation accuracy.

✅ 9. Evaluation

- Loads the saved model weights from disk.
- Uses the test data to generate predictions.
- Evaluates performance using a **confusion matrix**.

🌌 10. Feature Extraction & Visualization

- A secondary model is created to extract **512-dimensional features** from the last dense layer.
- Uses **t-SNE** to reduce feature vectors to 2D.
- Plots 2D feature space with labels to visualize class separability.

🧪 Demonstration Plan

Use this plan as a structure for your live or recorded demo:

1. **Introduction**: Briefly explain ASL and the goal of the project.
2. **Environment Setup**: Walk through mounting Google Drive, loading Kaggle API key, and installing dependencies.
3. **Preprocessing**: Show how metadata is created from directory structure.
4. **Model Building**:
    - Load VGG16
    - Add dense layers
    - Display `model.summary()`
5. **Training**: Explain loss, optimizer, and show the live training output.
6. **Evaluation**: Show the final test accuracy or confusion matrix.
7. **Visualization**: Display the t-SNE plot to explain feature embedding.