📦 Cell 1 – Download Dataset from Kaggle

```
!pip install -q kaggle

from google.colab import files
files.upload()  # Upload kaggle.json

!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

# Download flowers dataset
!kaggle datasets download -d alxmamaev/flowers-recognition
!unzip -q flowers-recognition.zip -d flower_data
```

Choose Files kaggle.json

- **kaggle.json**(application/json) - 68 bytes, last modified: 6/16/2025 - 100% done

```
Saving kaggle.json to kaggle.json
Dataset URL: https://www.kaggle.com/datasets/alxmamaev/flowers-recognition
License(s): unknown
Downloading flowers-recognition.zip to /content
 55% 124M/225M [00:00<00:00, 1.29GB/s]
100% 225M/225M [00:00<00:00, 843MB/s]
```

🧩 Cell 2 – Prepare Train/Test Split

```
import os
import shutil
import random

original_data_dir = "flower_data/flowers"
base_dir = "flower_data/split_data"
train_dir = os.path.join(base_dir, "train")
test_dir = os.path.join(base_dir, "test")

if not os.path.exists(train_dir):
    os.makedirs(train_dir)
    os.makedirs(test_dir)

    class_names = os.listdir(original_data_dir)

    for class_name in class_names:
        class_path = os.path.join(original_data_dir, class_name)
        if not os.path.isdir(class_path): continue
        images = os.listdir(class_path)
        random.shuffle(images)

        split_idx = int(0.8 * len(images))
        train_images = images[:split_idx]
        test_images = images[split_idx:]

        os.makedirs(os.path.join(train_dir, class_name), exist_ok=True)
        os.makedirs(os.path.join(test_dir, class_name), exist_ok=True)

        for img in train_images:
            shutil.copy(os.path.join(class_path, img), os.path.join(train_dir, class_name, img))
        for img in test_images:
            shutil.copy(os.path.join(class_path, img), os.path.join(test_dir, class_name, img))
```

🧩 Cell 3 – Image Data Generators

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

img_size = 224
batch_size = 32

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    zoom_range=0.3,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    horizontal_flip=True,
    brightness_range=[0.7, 1.4],
    fill_mode='nearest'
)

val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_size, img_size),
    batch_size=batch_size,
    class_mode='categorical'
)

val_generator = val_datagen.flow_from_directory(
    test_dir,
    target_size=(img_size, img_size),
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)
```

```
Found 3452 images belonging to 5 classes.
Found 865 images belonging to 5 classes.
```

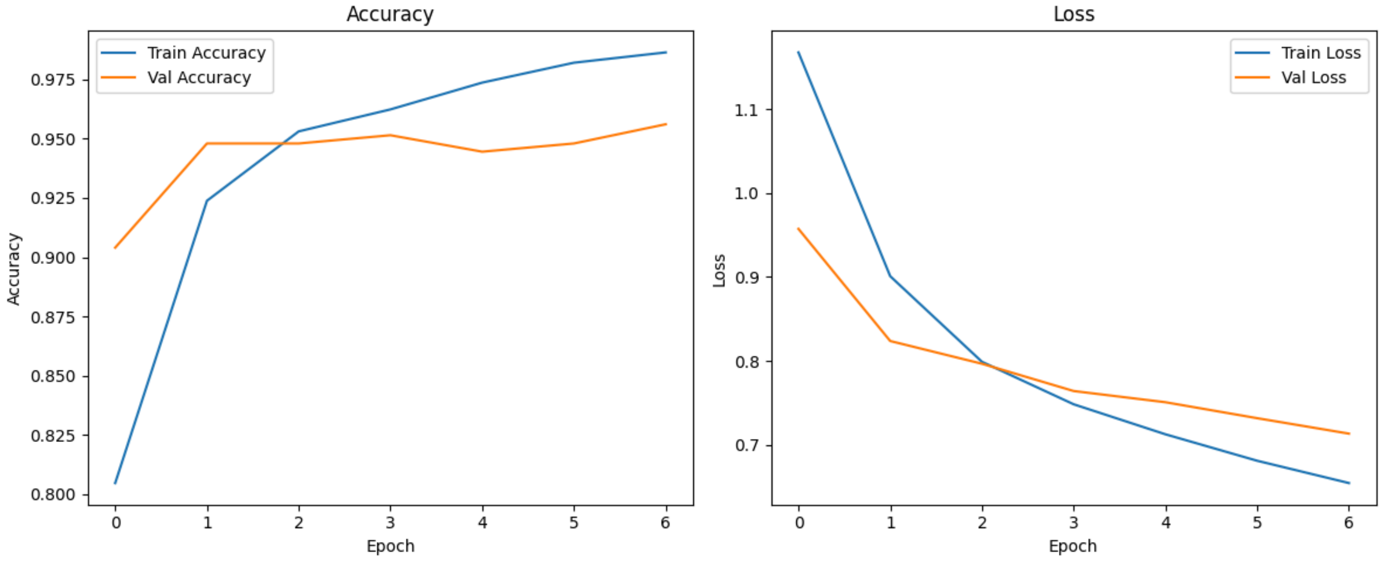
🧩 Cell 4 – Build & Train DenseNet201

```
from tensorflow.keras.applications import DenseNet201
from tensorflow.keras import layers, models, optimizers, regularizers
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.losses import CategoricalCrossentropy

# Load DenseNet201
base_model = DenseNet201(weights='imagenet', include_top=False, input_shape=(img_size, img_size, 3))
base_model.trainable = True

# Freeze early layers
for layer in base_model.layers[:200]:
    layer.trainable = False

# Classification head
num_classes = train_generator.num_classes
model = models.Sequential([
    base_model,
    layers.BatchNormalization(),
    layers.GlobalAveragePooling2D(),
    layers.Dense(512, activation='relu', kernel_regularizer=regularizers.l2(0.0005)),
    layers.Dropout(0.4),
    layers.Dense(num_classes, activation='softmax')
])

# Compile
model.compile(
    optimizer=optimizers.Adam(learning_rate=1e-4),
    loss=CategoricalCrossentropy(label_smoothing=0.1),
    metrics=['accuracy']
)

# Callbacks
early_stop = EarlyStopping(monitor='val_accuracy', patience=4, restore_best_weights=True)
checkpoint = ModelCheckpoint('flowers_densenet201_best.h5', save_best_only=True, monitor='val_accuracy')
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.3, patience=2, verbose=1, min_lr=1e-6)

# Train
epochs = 7
history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=epochs,
    callbacks=[early_stop, checkpoint, reduce_lr]
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet201_weights_tf_dim_ordering_tf_kerne
74836368/74836368 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class
  self._warn_if_super_not_called()
Epoch 1/7
108/108 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.6961 - loss: 1.3672WARNING:absl:You are saving your model as an HDF5 file via `moc
108/108 ━━━━━━━━━━━━━━━━━━━━ 604s 3s/step - accuracy: 0.6971 - loss: 1.3654 - val_accuracy: 0.9040 - val_loss: 0.9572 - learning_rate: 1
Epoch 2/7
108/108 ━━━━━━━━━━━━━━━━━━━━ 0s 574ms/step - accuracy: 0.9289 - loss: 0.9079WARNING:absl:You are saving your model as an HDF5 file via `
108/108 ━━━━━━━━━━━━━━━━━━━━ 68s 627ms/step - accuracy: 0.9288 - loss: 0.9078 - val_accuracy: 0.9480 - val_loss: 0.8236 - learning_rate:
Epoch 3/7
108/108 ━━━━━━━━━━━━━━━━━━━━ 64s 587ms/step - accuracy: 0.9560 - loss: 0.8072 - val_accuracy: 0.9480 - val_loss: 0.7966 - learning_rate:
Epoch 4/7
108/108 ━━━━━━━━━━━━━━━━━━━━ 0s 556ms/step - accuracy: 0.9670 - loss: 0.7442WARNING:absl:You are saving your model as an HDF5 file via `
108/108 ━━━━━━━━━━━━━━━━━━━━ 66s 608ms/step - accuracy: 0.9669 - loss: 0.7442 - val_accuracy: 0.9514 - val_loss: 0.7640 - learning_rate:
Epoch 5/7
108/108 ━━━━━━━━━━━━━━━━━━━━ 64s 592ms/step - accuracy: 0.9771 - loss: 0.7116 - val_accuracy: 0.9445 - val_loss: 0.7507 - learning_rate:
Epoch 6/7
108/108 ━━━━━━━━━━━━━━━━━━━━ 81s 587ms/step - accuracy: 0.9792 - loss: 0.6884 - val_accuracy: 0.9480 - val_loss: 0.7317 - learning_rate:
Epoch 7/7
108/108 ━━━━━━━━━━━━━━━━━━━━ 0s 551ms/step - accuracy: 0.9864 - loss: 0.6579WARNING:absl:You are saving your model as an HDF5 file via `
108/108 ━━━━━━━━━━━━━━━━━━━━ 67s 615ms/step - accuracy: 0.9864 - loss: 0.6579 - val_accuracy: 0.9561 - val_loss: 0.7133 - learning_rate:
```

🧩 Cell 5 – Plot Accuracy & Loss

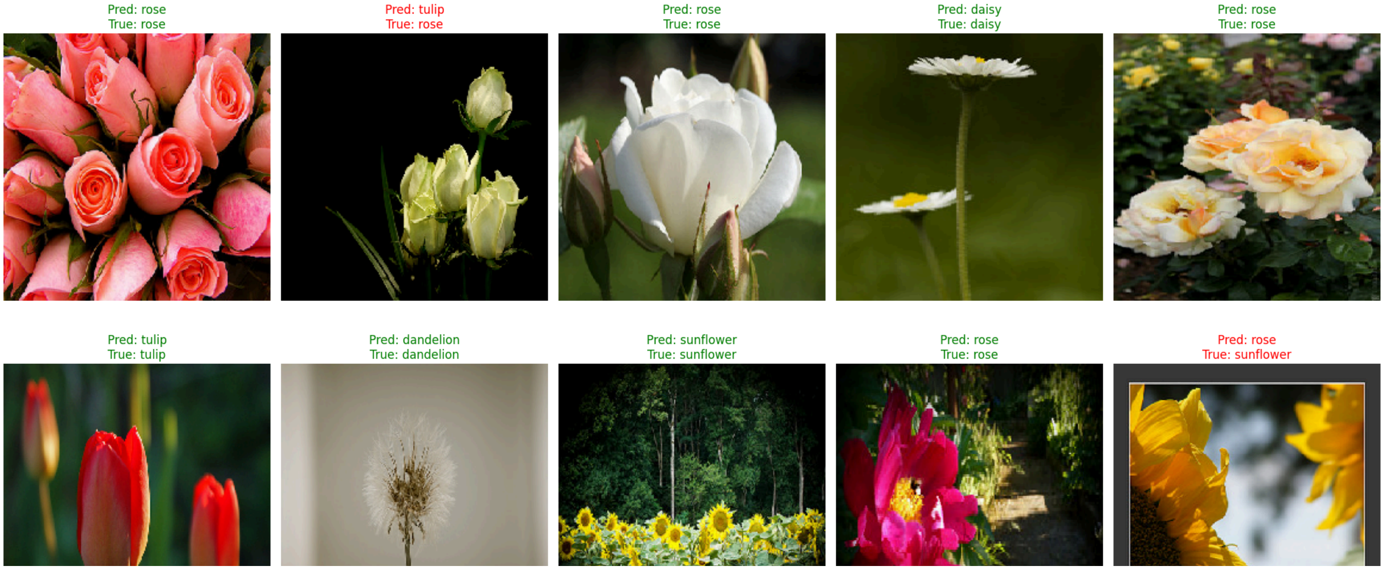
```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.title('Accuracy')
plt.xlabel('Epoch'); plt.ylabel('Accuracy'); plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Loss')
plt.xlabel('Epoch'); plt.ylabel('Loss'); plt.legend()

plt.tight_layout()
plt.show()
```



```
import os
from tensorflow.keras.preprocessing.image import load_img, img_to_array

# Path to validation directory
```

```
val_path = test_dir  # This was set earlier as: test_dir = "flower_data/split_data/test"
class_labels = sorted(os.listdir(val_path))

plt.figure(figsize=(20, 10))

i = 0
while i < 10:
    # Pick a random class
    class_name = random.choice(class_labels)
    class_dir = os.path.join(val_path, class_name)
    # Pick a random image from that class
    img_name = random.choice(os.listdir(class_dir))
    img_path = os.path.join(class_dir, img_name)

    # Load and preprocess image
    img = load_img(img_path, target_size=(img_size, img_size))
    img_array = img_to_array(img) / 255.0
    img_array_exp = np.expand_dims(img_array, axis=0)

    # Predict
    pred = model.predict(img_array_exp)
    pred_class = class_labels[np.argmax(pred)]

    # Display
    plt.subplot(2, 5, i + 1)
    plt.imshow(img)
    plt.axis('off')
    color = 'green' if pred_class == class_name else 'red'
    plt.title(f"Pred: {pred_class}\nTrue: {class_name}", color=color)

    i += 1

plt.tight_layout()
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 19s 19s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
```



Pred: rose
True: rose

Pred: tulip
True: rose

Pred: rose
True: rose

Pred: daisy
True: daisy

Pred: rose
True: rose

Pred: tulip
True: tulip

Pred: dandelion
True: dandelion

Pred: sunflower
True: sunflower

Pred: rose
True: rose

Pred: rose
True: sunflower