Image classification model using dataset Kaggel

Install Required Libraries and Authenticate Kaggle python

```
!pip install -q kaggle

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

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

```
Choose files kaggle.json
• kaggle.json(application/json) - 63 bytes, last modified: 15/06/2025 - 100% done
Saving kaggle.json to kaggle.json
```

Download the dataset from Kaggle

```
!kaggle datasets download -d mahmoudreda55/satellite-image-classification

# Unzip the downloaded dataset
!unzip -q satellite-image-classification.zip -d satellite_data
```

```
Dataset URL: https://www.kaggle.com/datasets/mahmoudreda55/satellite-image-classification
License(s): copyright-authors
Downloading satellite-image-classification.zip to /content
  0% 0.00/21.8M [00:00<?, ?B/s]
100% 21.8M/21.8M [00:00<00:00, 275MB/s]
```

Data Augmentation & Generators python

```
import os
import shutil
import random
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.xception import preprocess_input

# Define source and target directories
original_data_dir = "satellite_data/data"
base_dir = "satellite_data/split_data"
train_dir = os.path.join(base_dir, "train")
test_dir = os.path.join(base_dir, "test")

# Create split folders if not exist
if not os.path.exists(base_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)
        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))
        os.makedirs(os.path.join(test_dir, class_name))

        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))

# Define data generators
img_size = 224
batch_size = 32

train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    rotation_range=30,
    zoom_range=0.2,
```

```
    horizontal_flip=True,
    vertical_flip=True
)

test_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)

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

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

```
Found 4504 images belonging to 4 classes.
Found 1127 images belonging to 4 classes.
```

Import Librariest

```
from tensorflow.keras.applications import Xception
from tensorflow.keras import layers, models, optimizers
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
```

Load Base Model

```
# Step 1: Load Xception base model
base_model = Xception(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
base_model.trainable = False  # Freeze base model for transfer learning
```

Build Full Model

```
# Step 2: Build the full classification model
model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(512, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(train_generator.num_classes, activation='softmax')
])
```

Compile Model

```
model.compile(optimizer=optimizers.Adam(learning_rate=0.0001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Train the model

```
epochs = 3

early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
checkpoint = ModelCheckpoint('xception_best_model.h5', save_best_only=True)

history = model.fit(
    train_generator,
    validation_data=test_generator,
    epochs=epochs,
    callbacks=[early_stop, checkpoint]
)
```

```
Epoch 1/3
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 8s/step - accuracy: 0.7527 - loss: 0.8759WARNING:absl:You are saving your model as an HDF5 file via
141/141 ━━━━━━━━━━━━━━━━━━━━ 1483s 10s/step - accuracy: 0.7535 - loss: 0.8739 - val_accuracy: 0.9547 - val_loss: 0.3039
Epoch 2/3
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 8s/step - accuracy: 0.9498 - loss: 0.2400WARNING:absl:You are saving your model as an HDF5 file via
141/141 ━━━━━━━━━━━━━━━━━━━━ 1509s 10s/step - accuracy: 0.9498 - loss: 0.2398 - val_accuracy: 0.9672 - val_loss: 0.1709
Epoch 3/3
141/141 ━━━━━━━━━━━━━━━━━━━━ 0s 8s/step - accuracy: 0.9680 - loss: 0.1405WARNING:absl:You are saving your model as an HDF5 file via
```

```
141/141 ━━━━━━━━━━━━━━━━━━━━ 1449s 10s/step - accuracy: 0.9680 - loss: 0.1405 - val_accuracy: 0.9716 - val_loss: 0.1303
```

Plot Accuracy & Loss Curves

```
import matplotlib.pyplot as plt

# Accuracy plot
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy', marker='o')
plt.plot(history.history['val_accuracy'], label='Val Accuracy', marker='o')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

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

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