```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers, models
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical
```

```
pip install keras
```

```
     Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (2.15.0)
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
     Mounted at /content/drive
```

Load Dataset

```
# @title Load Dataset
def load_data(dataset_dir, img_size=(224, 224)):
  images=[]
  labels=[]

  classes=os.listdir(dataset_dir)
  label_encoder=LabelEncoder()
  for class_name in classes:
    class_dir=os.path.join(dataset_dir, class_name)

    if not os.path.isdir(class_dir):
      continue
    for img_file in os.listdir(class_dir):
      img_path=os.path.join(class_dir, img_file)

      img=cv2.imread(img_path)
      img=cv2.resize(img, img_size)
      img=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

      images.append(img)
      labels.append(class_name)
  labels_encoded = label_encoder.fit_transform(labels)
  labels_one_hot = to_categorical(labels_encoded)

  return np.array(images), np.array(labels_one_hot)
```

```
dataset_dir='/content/drive/MyDrive/db/Indian Food Images'
images, labels=load_data(dataset_dir)

print("Number of images:", len(images))
print("Number of labels:", len(labels))
```

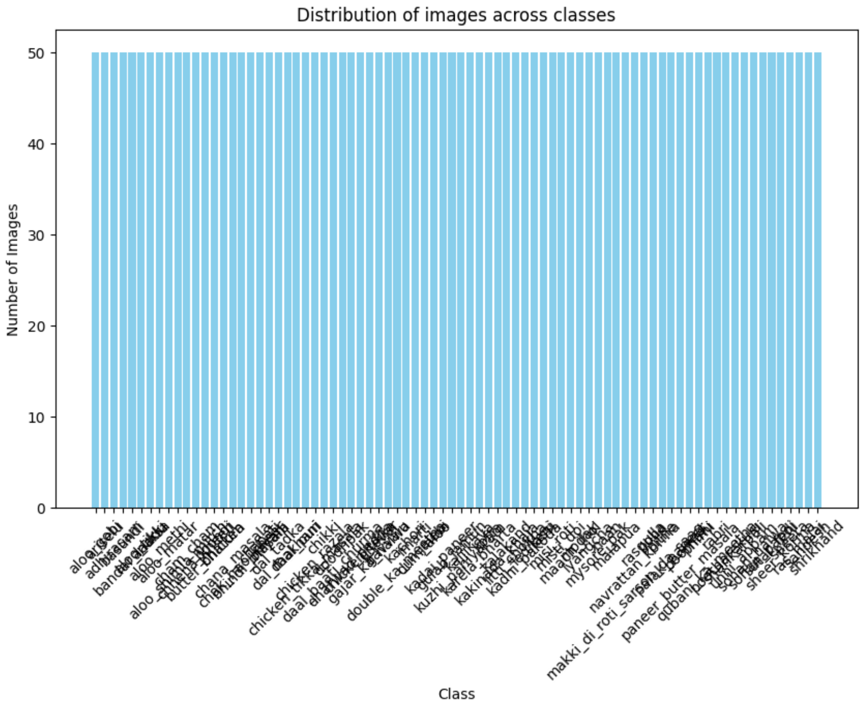
```
     Number of images: 4000
     Number of labels: 4000
```

Data exploration

```
# @title Data exploration
classes=os.listdir(dataset_dir)

class_counts={}
for class_name in classes:
  class_dir=os.path.join(dataset_dir, class_name)
  num_images=len(os.listdir(class_dir))
  class_counts[class_name]=num_images
```

```
plt.figure(figsize=(10, 6))
plt.bar(class_counts.keys(), class_counts.values(), color='skyblue')
plt.xlabel('Class')
plt.ylabel('Number of Images')
plt.title('Distribution of images across classes')
plt.xticks(rotation=45)
plt.show()
```



```
class_df=pd.DataFrame(list(class_counts.items()), columns=['Class', 'Number of images'])
class_df
```

	Class	Number of images
0	aloo_gobi	50
1	ariselu	50
2	adhirasam	50
3	basundi	50
4	bandar_laddu	50
...	...	...
75	sheer_korma	50
76	sheera	50
77	ras_malai	50
78	sandesh	50
79	shrikhand	50

80 rows × 2 columns

Next steps: Generate code with class_df View recommended plots

```
total_imgs=sum(class_counts.values())
print("Total no of images:", total_imgs)
print("No of classes:", len(classes))
print("Avg imgs per class:", total_imgs/len(classes))
```

```
Total no of images: 4000
No of classes: 80
Avg imgs per class: 50.0
```

Preprocessing

```
# @title Preprocessing
preprocessed_images=[]
preprocessed_labels=[]

for img, label in zip(images, labels):
  img=img.astype(np.float32)/255.0

  preprocessed_images.append(img)
  preprocessed_labels.append(label)

preprocessed_images=np.array(preprocessed_images)
preprocessed_labels=np.array(preprocessed_labels)
print("Preprocessed images shape:", preprocessed_images.shape)
print("Preprocessed labels shape:", preprocessed_labels.shape)
```

```
Preprocessed images shape: (4000, 224, 224, 3)
Preprocessed labels shape: (4000, 80)
```

Transfer Learning

```
# @title Transfer Learning
def create_model(input_shape, num_classes):
  model=models.Sequential([
      layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
      layers.MaxPooling2D((2, 2)),
      layers.Conv2D(64, (3, 3), activation='relu'),
      layers.MaxPooling2D((2, 2)),
      layers.Conv2D(128, (3, 3), activation='relu'),
      layers.MaxPooling2D((2, 2)),

      layers.Flatten(),

      layers.Dense(128, activation='relu'),
      layers.Dense(num_classes, activation='softmax')
  ])

  model.compile(optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy'])

  return model

input_shape=(224, 224, 3)
num_classes=80
model=create_model(input_shape, num_classes)

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 32)      896

 max_pooling2d (MaxPooling2  (None, 111, 111, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 109, 109, 64)      18496

 max_pooling2d_1 (MaxPoolin  (None, 54, 54, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 52, 52, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 26, 26, 128)       0
 g2D)

 flatten (Flatten)           (None, 86528)             0

 dense (Dense)               (None, 128)               11075712

 dense_1 (Dense)             (None, 80)                10320

=================================================================
Total params: 11179280 (42.65 MB)
Trainable params: 11179280 (42.65 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Model Training

```
# @title Model Training
def build_transfer_learning_model(input_shape, num_classes):
  base_model=VGG16(weights='imagenet', include_top=False, input_shape=input_shape)

  for layer in base_model.layers:
    layers.trainable=False

  x=Flatten()(base_model.output)
  x=Dense(512, activation='relu')(x)
  x=Dropout(0.5)(x)
  output=Dense(num_classes, activation='softmax')(x)

  model=Model(inputs=base_model.input, outputs=output)

  model.compile(optimizer=Adam(),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
  return model

transfer_learning_model=build_transfer_learning_model(input_shape, num_classes)

transfer_learning_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels
58889256/58889256 [==============================] - 4s 0us/step
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten_1 (Flatten)         (None, 25088)             0

 dense_2 (Dense)             (None, 512)               12845568

 dropout (Dropout)           (None, 512)               0

 dense_3 (Dense)             (None, 80)                41040

=================================================================
Total params: 27601296 (105.29 MB)
Trainable params: 27601296 (105.29 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Evaluation

```
# @title Evaluation
x_data, y_data=load_data(dataset_dir)

x_train, x_test, y_train, y_test=train_test_split(x_data, y_data, test_size=0.2, random_state=42)
print(x_train.shape)
print(y_train.shape)

epochs=10
batch_size=32

history=transfer_learning_model.fit(x_train, y_train,
                                    epochs=epochs,
                                    batch_size=batch_size,
                                    validation_data=(x_test, y_test))
```

```
(3200, 224, 224, 3)
(3200, 80)
Epoch 1/10
100/100 [==============================] - 71s 435ms/step - loss: 5.7060 - accuracy: 0.0134 - val_loss: 4.3840 - val_accuracy: 0.011
Epoch 2/10
100/100 [==============================] - 44s 443ms/step - loss: 4.3820 - accuracy: 0.0109 - val_loss: 4.3856 - val_accuracy: 0.008
Epoch 3/10
100/100 [==============================] - 42s 415ms/step - loss: 4.3809 - accuracy: 0.0094 - val_loss: 4.3877 - val_accuracy: 0.005
Epoch 4/10
100/100 [==============================] - 44s 441ms/step - loss: 4.6207 - accuracy: 0.0131 - val_loss: 4.4374 - val_accuracy: 0.011
Epoch 5/10
100/100 [==============================] - 44s 440ms/step - loss: 4.4212 - accuracy: 0.0128 - val_loss: 4.3912 - val_accuracy: 0.003
Epoch 6/10
100/100 [==============================] - 42s 421ms/step - loss: 4.3752 - accuracy: 0.0181 - val_loss: 4.3934 - val_accuracy: 0.011
Epoch 7/10
100/100 [==============================] - 42s 422ms/step - loss: 4.3654 - accuracy: 0.0188 - val_loss: 4.3935 - val_accuracy: 0.005
Epoch 8/10
100/100 [==============================] - 42s 420ms/step - loss: 4.3560 - accuracy: 0.0213 - val_loss: 4.3962 - val_accuracy: 0.003
Epoch 9/10
100/100 [==============================] - 44s 438ms/step - loss: 4.3716 - accuracy: 0.0172 - val_loss: 4.3972 - val_accuracy: 0.003
Epoch 10/10
100/100 [==============================] - 42s 418ms/step - loss: 4.3522 - accuracy: 0.0206 - val_loss: 4.3992 - val_accuracy: 0.005
```

Start coding or generate with AI.