Import and unzip the dataset

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

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
!unzip "/content/drive/MyDrive/Digital Naturalist Dataset (1).zip"
```

```
Archive:  /content/drive/MyDrive/Digital Naturalist Dataset (1).zip
   creating: Digital Naturalist Dataset/
   creating: Digital Naturalist Dataset/Bird/
   creating: Digital Naturalist Dataset/Bird/Great Indian Bustard
Bird/
  inflating: Digital Naturalist Dataset/Bird/Great Indian Bustard
Bird/download (1).jpg
  inflating: Digital Naturalist Dataset/Bird/Great Indian Bustard
Bird/download (10).jpg
  inflating: Digital Naturalist Dataset/Bird/Great Indian Bustard
Bird/download (11).jpg
  inflating: Digital Naturalist Dataset/Bird/Great Indian Bustard
Bird/download (2).jpg
  inflating: Digital Naturalist Dataset/Bird/Great Indian Bustard
Bird/download (3).jpg
  inflating: Digital Naturalist Dataset/Bird/Great Indian Bustard
Bird/download (4).jpg
  inflating: Digital Naturalist Dataset/Bird/Great Indian Bustard
Bird/download (5).jpg
  inflating: Digital Naturalist Dataset/Bird/Great Indian Bustard
Bird/download (6).jpg
  inflating: Digital Naturalist Dataset/Bird/Great Indian Bustard
Bird/download (7).jpg
  inflating: Digital Naturalist Dataset/Bird/Great Indian Bustard
Bird/download (8).jpg
  inflating: Digital Naturalist Dataset/Bird/Great Indian Bustard
Bird/download (9).jpg
  inflating: Digital Naturalist Dataset/Bird/Great Indian Bustard
Bird/download.jpg
 extracting: Digital Naturalist Dataset/Bird/Great Indian Bustard
Bird/greatindianbustard-kRDC--621x414@LiveMint.webp
  inflating: Digital Naturalist Dataset/Bird/Great Indian Bustard
Bird/images (1).jpg
  inflating: Digital Naturalist Dataset/Bird/Great Indian Bustard
Bird/images (2).jpg
  inflating: Digital Naturalist Dataset/Bird/Great Indian Bustard
Bird/images (3).jpg
  inflating: Digital Naturalist Dataset/Bird/Great Indian Bustard
Bird/images (4).jpg
```

```
  inflating: Digital Naturalist Dataset/Bird/Great Indian Bustard
Bird/images (5).jpg
  inflating: Digital Naturalist Dataset/Bird/Great Indian Bustard
Bird/images (6).jpg
  inflating: Digital Naturalist Dataset/Bird/Great Indian Bustard
Bird/images (7).jpg
  inflating: Digital Naturalist Dataset/Bird/Great Indian Bustard
Bird/images (8).jpg
  inflating: Digital Naturalist Dataset/Bird/Great Indian Bustard
Bird/images.jpg
   creating: Digital Naturalist Dataset/Bird/Spoon Billed Sandpiper
Bird/
  inflating: Digital Naturalist Dataset/Bird/Spoon Billed Sandpiper
Bird/download (1).jpg
  inflating: Digital Naturalist Dataset/Bird/Spoon Billed Sandpiper
Bird/download (12).jpg
  inflating: Digital Naturalist Dataset/Bird/Spoon Billed Sandpiper
Bird/download (2).jpg
  inflating: Digital Naturalist Dataset/Bird/Spoon Billed Sandpiper
Bird/download (3).jpg
  inflating: Digital Naturalist Dataset/Bird/Spoon Billed Sandpiper
Bird/download (4).jpg
  inflating: Digital Naturalist Dataset/Bird/Spoon Billed Sandpiper
Bird/download (5).jpg
  inflating: Digital Naturalist Dataset/Bird/Spoon Billed Sandpiper
Bird/download (6).jpg
  inflating: Digital Naturalist Dataset/Bird/Spoon Billed Sandpiper
Bird/download (7).jpg
  inflating: Digital Naturalist Dataset/Bird/Spoon Billed Sandpiper
Bird/download.jpg
  inflating: Digital Naturalist Dataset/Bird/Spoon Billed Sandpiper
Bird/images (1).jpg
  inflating: Digital Naturalist Dataset/Bird/Spoon Billed Sandpiper
Bird/images (10).jpg
  inflating: Digital Naturalist Dataset/Bird/Spoon Billed Sandpiper
Bird/images (11).jpg
  inflating: Digital Naturalist Dataset/Bird/Spoon Billed Sandpiper
Bird/images (2).jpg
  inflating: Digital Naturalist Dataset/Bird/Spoon Billed Sandpiper
Bird/images (3).jpg
  inflating: Digital Naturalist Dataset/Bird/Spoon Billed Sandpiper
Bird/images (4).jpg
  inflating: Digital Naturalist Dataset/Bird/Spoon Billed Sandpiper
Bird/images (5).jpg
  inflating: Digital Naturalist Dataset/Bird/Spoon Billed Sandpiper
Bird/images (6).jpg
  inflating: Digital Naturalist Dataset/Bird/Spoon Billed Sandpiper
Bird/images (7).jpg
  inflating: Digital Naturalist Dataset/Bird/Spoon Billed Sandpiper
Bird/images (8).jpg
```

```
  inflating: Digital Naturalist Dataset/Bird/Spoon Billed Sandpiper
Bird/images (9).jpg
  inflating: Digital Naturalist Dataset/Bird/Spoon Billed Sandpiper
Bird/images.jpg
   creating: Digital Naturalist Dataset/Flower/
   creating: Digital Naturalist Dataset/Flower/Corpse Flower/
  inflating: Digital Naturalist Dataset/Flower/Corpse Flower/download
(1).jpg
  inflating: Digital Naturalist Dataset/Flower/Corpse Flower/download
(11).jpg
  inflating: Digital Naturalist Dataset/Flower/Corpse Flower/download
(2).jpg
  inflating: Digital Naturalist Dataset/Flower/Corpse Flower/download
(3).jpg
  inflating: Digital Naturalist Dataset/Flower/Corpse Flower/download
(4).jpg
  inflating: Digital Naturalist Dataset/Flower/Corpse Flower/download
(5).jpg
  inflating: Digital Naturalist Dataset/Flower/Corpse
Flower/download.jpg
  inflating: Digital Naturalist Dataset/Flower/Corpse Flower/images
(1).jpg
  inflating: Digital Naturalist Dataset/Flower/Corpse Flower/images
(10).jpg
  inflating: Digital Naturalist Dataset/Flower/Corpse Flower/images
(11).jpg
  inflating: Digital Naturalist Dataset/Flower/Corpse Flower/images
(12).jpg
  inflating: Digital Naturalist Dataset/Flower/Corpse Flower/images
(13).jpg
  inflating: Digital Naturalist Dataset/Flower/Corpse Flower/images
(14).jpg
  inflating: Digital Naturalist Dataset/Flower/Corpse Flower/images
(15).jpg
  inflating: Digital Naturalist Dataset/Flower/Corpse Flower/images
(16).jpg
 extracting: Digital Naturalist Dataset/Flower/Corpse Flower/images
(17).jpg
  inflating: Digital Naturalist Dataset/Flower/Corpse Flower/images
(2).jpg
  inflating: Digital Naturalist Dataset/Flower/Corpse Flower/images
(3).jpg
  inflating: Digital Naturalist Dataset/Flower/Corpse Flower/images
(4).jpg
  inflating: Digital Naturalist Dataset/Flower/Corpse Flower/images
(5).jpg
  inflating: Digital Naturalist Dataset/Flower/Corpse Flower/images
(6).jpg
  inflating: Digital Naturalist Dataset/Flower/Corpse Flower/images
(7).jpg
```

```
  inflating: Digital Naturalist Dataset/Flower/Corpse Flower/images
(8).jpg
  inflating: Digital Naturalist Dataset/Flower/Corpse Flower/images
(9).jpg
 extracting: Digital Naturalist Dataset/Flower/Corpse
Flower/images.jpg
   creating: Digital Naturalist Dataset/Flower/Lady Slipper Orchid
Flower/
  inflating: Digital Naturalist Dataset/Flower/Lady Slipper Orchid
Flower/download (1).jpg
  inflating: Digital Naturalist Dataset/Flower/Lady Slipper Orchid
Flower/download (10).jpg
  inflating: Digital Naturalist Dataset/Flower/Lady Slipper Orchid
Flower/download (2).jpg
  inflating: Digital Naturalist Dataset/Flower/Lady Slipper Orchid
Flower/download (3).jpg
  inflating: Digital Naturalist Dataset/Flower/Lady Slipper Orchid
Flower/download (4).jpg
  inflating: Digital Naturalist Dataset/Flower/Lady Slipper Orchid
Flower/download (5).jpg
  inflating: Digital Naturalist Dataset/Flower/Lady Slipper Orchid
Flower/download (6).jpg
  inflating: Digital Naturalist Dataset/Flower/Lady Slipper Orchid
Flower/download (7).jpg
  inflating: Digital Naturalist Dataset/Flower/Lady Slipper Orchid
Flower/download (8).jpg
  inflating: Digital Naturalist Dataset/Flower/Lady Slipper Orchid
Flower/download (9).jpg
  inflating: Digital Naturalist Dataset/Flower/Lady Slipper Orchid
Flower/download.jpg
  inflating: Digital Naturalist Dataset/Flower/Lady Slipper Orchid
Flower/images (1).jpg
  inflating: Digital Naturalist Dataset/Flower/Lady Slipper Orchid
Flower/images (10).jpg
  inflating: Digital Naturalist Dataset/Flower/Lady Slipper Orchid
Flower/images (11).jpg
  inflating: Digital Naturalist Dataset/Flower/Lady Slipper Orchid
Flower/images (12).jpg
  inflating: Digital Naturalist Dataset/Flower/Lady Slipper Orchid
Flower/images (2).jpg
  inflating: Digital Naturalist Dataset/Flower/Lady Slipper Orchid
Flower/images (3).jpg
  inflating: Digital Naturalist Dataset/Flower/Lady Slipper Orchid
Flower/images (4).jpg
  inflating: Digital Naturalist Dataset/Flower/Lady Slipper Orchid
Flower/images (5).jpg
  inflating: Digital Naturalist Dataset/Flower/Lady Slipper Orchid
Flower/images (6).jpg
  inflating: Digital Naturalist Dataset/Flower/Lady Slipper Orchid
Flower/images (7).jpg
```

```
  inflating: Digital Naturalist Dataset/Flower/Lady Slipper Orchid
Flower/images (8).jpg
  inflating: Digital Naturalist Dataset/Flower/Lady Slipper Orchid
Flower/images (9).jpg
  inflating: Digital Naturalist Dataset/Flower/Lady Slipper Orchid
Flower/images.jpg
   creating: Digital Naturalist Dataset/Mammal/
   creating: Digital Naturalist Dataset/Mammal/Pangolin Mammal/
  inflating: Digital Naturalist Dataset/Mammal/Pangolin
Mammal/download (1).jpg
  inflating: Digital Naturalist Dataset/Mammal/Pangolin
Mammal/download (2).jpg
  inflating: Digital Naturalist Dataset/Mammal/Pangolin
Mammal/download (3).jpg
  inflating: Digital Naturalist Dataset/Mammal/Pangolin
Mammal/download (4).jpg
 extracting: Digital Naturalist Dataset/Mammal/Pangolin
Mammal/download (5).jpg
  inflating: Digital Naturalist Dataset/Mammal/Pangolin
Mammal/download (6).jpg
  inflating: Digital Naturalist Dataset/Mammal/Pangolin
Mammal/download (8).jpg
  inflating: Digital Naturalist Dataset/Mammal/Pangolin
Mammal/download.jpg
  inflating: Digital Naturalist Dataset/Mammal/Pangolin Mammal/images
(1).jpg
  inflating: Digital Naturalist Dataset/Mammal/Pangolin Mammal/images
(10).jpg
  inflating: Digital Naturalist Dataset/Mammal/Pangolin Mammal/images
(11).jpg
  inflating: Digital Naturalist Dataset/Mammal/Pangolin Mammal/images
(12).jpg
  inflating: Digital Naturalist Dataset/Mammal/Pangolin Mammal/images
(13).jpg
  inflating: Digital Naturalist Dataset/Mammal/Pangolin Mammal/images
(14).jpg
  inflating: Digital Naturalist Dataset/Mammal/Pangolin Mammal/images
(15).jpg
  inflating: Digital Naturalist Dataset/Mammal/Pangolin Mammal/images
(16).jpg
  inflating: Digital Naturalist Dataset/Mammal/Pangolin Mammal/images
(2).jpg
  inflating: Digital Naturalist Dataset/Mammal/Pangolin Mammal/images
(3).jpg
  inflating: Digital Naturalist Dataset/Mammal/Pangolin Mammal/images
(4).jpg
  inflating: Digital Naturalist Dataset/Mammal/Pangolin Mammal/images
(5).jpg
  inflating: Digital Naturalist Dataset/Mammal/Pangolin Mammal/images
(6).jpg
```

```
  inflating: Digital Naturalist Dataset/Mammal/Pangolin Mammal/images
(7).jpg
  inflating: Digital Naturalist Dataset/Mammal/Pangolin Mammal/images
(8).jpg
  inflating: Digital Naturalist Dataset/Mammal/Pangolin Mammal/images
(9).jpg
  inflating: Digital Naturalist Dataset/Mammal/Pangolin
Mammal/images.jpg
   creating: Digital Naturalist Dataset/Mammal/Senenca White Deer
Mammal/
  inflating: Digital Naturalist Dataset/Mammal/Senenca White Deer
Mammal/download (1).jpg
  inflating: Digital Naturalist Dataset/Mammal/Senenca White Deer
Mammal/download (2).jpg
  inflating: Digital Naturalist Dataset/Mammal/Senenca White Deer
Mammal/download (3).jpg
  inflating: Digital Naturalist Dataset/Mammal/Senenca White Deer
Mammal/download (7).jpg
  inflating: Digital Naturalist Dataset/Mammal/Senenca White Deer
Mammal/download.jpg
  inflating: Digital Naturalist Dataset/Mammal/Senenca White Deer
Mammal/images (1).jpg
  inflating: Digital Naturalist Dataset/Mammal/Senenca White Deer
Mammal/images (10).jpg
  inflating: Digital Naturalist Dataset/Mammal/Senenca White Deer
Mammal/images (11).jpg
  inflating: Digital Naturalist Dataset/Mammal/Senenca White Deer
Mammal/images (12).jpg
  inflating: Digital Naturalist Dataset/Mammal/Senenca White Deer
Mammal/images (13).jpg
  inflating: Digital Naturalist Dataset/Mammal/Senenca White Deer
Mammal/images (14).jpg
  inflating: Digital Naturalist Dataset/Mammal/Senenca White Deer
Mammal/images (15).jpg
  inflating: Digital Naturalist Dataset/Mammal/Senenca White Deer
Mammal/images (16).jpg
  inflating: Digital Naturalist Dataset/Mammal/Senenca White Deer
Mammal/images (2).jpg
  inflating: Digital Naturalist Dataset/Mammal/Senenca White Deer
Mammal/images (3).jpg
  inflating: Digital Naturalist Dataset/Mammal/Senenca White Deer
Mammal/images (4).jpg
  inflating: Digital Naturalist Dataset/Mammal/Senenca White Deer
Mammal/images (5).jpg
  inflating: Digital Naturalist Dataset/Mammal/Senenca White Deer
Mammal/images (6).jpg
  inflating: Digital Naturalist Dataset/Mammal/Senenca White Deer
Mammal/images (7).jpg
  inflating: Digital Naturalist Dataset/Mammal/Senenca White Deer
Mammal/images (8).jpg
```

```
  inflating: Digital Naturalist Dataset/Mammal/Senenca White Deer
Mammal/images (9).jpg
  inflating: Digital Naturalist Dataset/Mammal/Senenca White Deer
Mammal/images.jpg
```

Image Preprocessing

1.Import The ImageDataGenerator Library

```
#import required lib
import numpy as np
import tensorflow as tf
import keras
import keras.backend as K
from keras.optimizers import SGD, Adam, Adagrad, RMSprop
from keras.applications import *
from keras.preprocessing import *
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPool2D, Flatten,
Activation, BatchNormalization, Dropout
from keras.utils.np_utils import to_categorical
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import glob
from PIL import Image
import os
from os import listdir
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

2.Configure ImageDataGenerator Class

```
#Creating augmentation on training variable
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range = 0.1,
                                   zoom_range=0.1,
                                   horizontal_flip=True)
# Creating augmentation on testing variable

test_datagen = ImageDataGenerator(rescale=1./255)
```

3.Apply ImageDataGenerator Functionality To Trainset And Testset

```
# Passing training data to train variable for mammal

xtrain = train_datagen.flow_from_directory('/content/Digital
Naturalist Dataset/Mammal',
                                           target_size=(224,224),
```

```
                                                class_mode='categorical',
                                                batch_size=10)
```

```
Found 47 images belonging to 2 classes.
```

```
# Passing testing data to test variable for mammal

xtest = test_datagen.flow_from_directory('/content/Digital Naturalist
Dataset/Mammal',
                                                target_size=(224,224),
                                                class_mode='categorical',
                                                batch_size=10)
```

```
Found 47 images belonging to 2 classes.
```

```
# Passing training data to train variable for birds

xtrain1 = train_datagen.flow_from_directory('/content/Digital
Naturalist Dataset/Bird',
                                                target_size=(224,224),
                                                class_mode='categorical',
                                                batch_size=10)
```

```
Found 42 images belonging to 2 classes.
```

```
# Passing testing data to test variable for birds

xtest1 = test_datagen.flow_from_directory('/content/Digital Naturalist
Dataset/Bird',
                                                target_size=(224,224),
                                                class_mode='categorical',
                                                batch_size=10)
```

```
Found 42 images belonging to 2 classes.
```

```
# Passing training data to train variable for flowers

xtrain2 = train_datagen.flow_from_directory('/content/Digital
Naturalist Dataset/Flower',
                                                target_size=(224,224),
                                                class_mode='categorical',
                                                batch_size=10)
```

```
Found 49 images belonging to 2 classes.
```

```
# Passing testing data to test variable for flowers

xtest2 = test_datagen.flow_from_directory('/content/Digital Naturalist
Dataset/Flower',
                                                target_size=(224,224),
                                                class_mode='categorical',
                                                batch_size=10)
```

```
Found 49 images belonging to 2 classes.
```

For Mammal

1.Importing The Model Building Libraries

```
import tensorflow as tf
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import
ImageDataGenerator,load_img
from tensorflow.keras.models import Sequential
import numpy as np
from glob import glob
```

2.Loading The Model

```
IMAGE_SIZE = [224, 224]

train_path = '/content/Digital Naturalist Dataset/Mammal'
valid_path = '/content/Digital Naturalist Dataset/Mammal'
```

3.Adding Flatten Layer

```
vgg16 = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet',
include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 0s 0us/step
```

```
for layer in vgg16.layers:
    layer.trainable = False

folders = glob('/content/Digital Naturalist Dataset/Mammal/*')
folders
```

```
['/content/Digital Naturalist Dataset/Mammal/Senenca White Deer
Mammal',
 '/content/Digital Naturalist Dataset/Mammal/Pangolin Mammal']

x = Flatten()(vgg16.output)
len(folders)
```

```
2
```

4. Adding Output Layer

```
prediction = Dense(len(folders), activation='softmax')(x)
```

5. Creating A Model Object

```
model = Model(inputs=vgg16.input, outputs=prediction)
model.summary()
```

```
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 (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 2)                 50178
```

```
=================================================================
Total params: 14,764,866
Trainable params: 50,178
Non-trainable params: 14,714,688
_________________________________________________________________
```

6. Configure The Learning Process

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

7. Train The Model

```
r = model.fit(
  xtrain,
  validation_data=xtest,
  epochs=10,
  steps_per_epoch=len(xtrain),
  validation_steps=len(xtest)
)
```

```
Epoch 1/10
5/5 [==============================] - 52s 11s/step - loss: 1.1269 -
accuracy: 0.5106 - val_loss: 0.3888 - val_accuracy: 0.7872
Epoch 2/10
5/5 [==============================] - 50s 11s/step - loss: 0.1790 -
accuracy: 0.9362 - val_loss: 0.1786 - val_accuracy: 0.9362
Epoch 3/10
5/5 [==============================] - 50s 11s/step - loss: 0.1163 -
accuracy: 0.9787 - val_loss: 0.0288 - val_accuracy: 1.0000
Epoch 4/10
5/5 [==============================] - 49s 11s/step - loss: 0.0743 -
accuracy: 0.9574 - val_loss: 0.0453 - val_accuracy: 0.9787
Epoch 5/10
5/5 [==============================] - 50s 11s/step - loss: 0.0264 -
accuracy: 1.0000 - val_loss: 0.0084 - val_accuracy: 1.0000
Epoch 6/10
5/5 [==============================] - 50s 11s/step - loss: 0.0137 -
accuracy: 1.0000 - val_loss: 0.0056 - val_accuracy: 1.0000
Epoch 7/10
5/5 [==============================] - 50s 11s/step - loss: 0.0102 -
accuracy: 1.0000 - val_loss: 0.0065 - val_accuracy: 1.0000
Epoch 8/10
5/5 [==============================] - 50s 11s/step - loss: 0.0109 -
accuracy: 1.0000 - val_loss: 0.0028 - val_accuracy: 1.0000
Epoch 9/10
5/5 [==============================] - 50s 11s/step - loss: 0.0040 -
accuracy: 1.0000 - val_loss: 0.0018 - val_accuracy: 1.0000
```

```
Epoch 10/10
5/5 [==============================] - 50s 11s/step - loss: 0.0051 -
accuracy: 1.0000 - val_loss: 0.0015 - val_accuracy: 1.0000
```

8. Save The Model

```
from tensorflow.keras.models import load_model

model.save('/content/Digital Naturalist Dataset/Mammal.h5')
```

9. Test The Model

```
from tensorflow.keras.models import load_model
import cv2
from skimage.transform import resize
model = load_model('/content/Digital Naturalist Dataset/Mammal.h5')
def detect(frame):
  img = cv2.resize(frame,(224,224))
  img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

  if(np.max(img)>1):
    img = img/255.0
  img = np.array([img])
  prediction = model.predict(img)
  label = ["Pangolin Mammal","Seneca White Deer Mammal"]
  preds = label[np.argmax(prediction)]
  return preds
import numpy as np
data = "/content/Digital Naturalist Dataset/Mammal/Pangolin
Mammal/download (1).jpg"
image = cv2.imread(data)
print(detect(image))
```

```
1/1 [==============================] - 1s 693ms/step
Pangolin Mammal
```

FOR Birds

MODEL BUILDING

1. Importing The Model Building Libraries

```
import tensorflow as tf
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import
ImageDataGenerator,load_img
```

```
from tensorflow.keras.models import Sequential
import numpy as np
from glob import glob
```

2. Loading The Model

```
IMAGE_SIZE = [224, 224]

train_path = '/content/Digital Naturalist Dataset/Bird'
valid_path = '/content/Digital Naturalist Dataset/Bird'
```

3. Adding Flatten Layer

```
vgg16 = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet',
include_top=False)

for layer in vgg16.layers:
    layer.trainable = False
folders = glob('/content/Digital Naturalist Dataset/Bird/*')
folders
```

```
['/content/Digital Naturalist Dataset/Bird/Spoon Billed Sandpiper
Bird',
 '/content/Digital Naturalist Dataset/Bird/Great Indian Bustard Bird']
```

```
x = Flatten()(vgg16.output)
len(folders)
```

```
2
```

4. Adding Output Layer

```
prediction = Dense(len(folders), activation='softmax')(x)
```

5. Creating A Model Object

```
model = Model(inputs=vgg16.input, outputs=prediction)
model.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (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_1 (Dense)             (None, 2)                 50178

=================================================================
Total params: 14,764,866
Trainable params: 50,178
Non-trainable params: 14,714,688
_________________________________________________________________
```

6. Configure The Learning Process

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

7. Train The Model

```
r = model.fit(
  xtrain1,
  validation_data=xtest1,
```

```
  epochs=10,
  steps_per_epoch=len(xtrain1),
  validation_steps=len(xtest1)
)
```

```
Epoch 1/10
5/5 [==============================] - 47s 10s/step - loss: 0.8016 -
accuracy: 0.5714 - val_loss: 0.3238 - val_accuracy: 0.8571
Epoch 2/10
5/5 [==============================] - 45s 10s/step - loss: 0.4111 -
accuracy: 0.8333 - val_loss: 0.1311 - val_accuracy: 0.9286
Epoch 3/10
5/5 [==============================] - 45s 10s/step - loss: 0.1637 -
accuracy: 0.9048 - val_loss: 0.0740 - val_accuracy: 0.9762
Epoch 4/10
5/5 [==============================] - 45s 10s/step - loss: 0.2625 -
accuracy: 0.9048 - val_loss: 0.0706 - val_accuracy: 0.9762
Epoch 5/10
5/5 [==============================] - 45s 10s/step - loss: 0.0353 -
accuracy: 1.0000 - val_loss: 0.0978 - val_accuracy: 0.9524
Epoch 6/10
5/5 [==============================] - 43s 9s/step - loss: 0.1180 -
accuracy: 0.9524 - val_loss: 0.0266 - val_accuracy: 1.0000
Epoch 7/10
5/5 [==============================] - 45s 10s/step - loss: 0.0244 -
accuracy: 1.0000 - val_loss: 0.0160 - val_accuracy: 1.0000
Epoch 8/10
5/5 [==============================] - 45s 10s/step - loss: 0.0821 -
accuracy: 0.9524 - val_loss: 0.0181 - val_accuracy: 1.0000
Epoch 9/10
5/5 [==============================] - 44s 9s/step - loss: 0.0339 -
accuracy: 1.0000 - val_loss: 0.0083 - val_accuracy: 1.0000
Epoch 10/10
5/5 [==============================] - 45s 10s/step - loss: 0.0479 -
accuracy: 1.0000 - val_loss: 0.0101 - val_accuracy: 1.0000
```

8. Save And Test The Model

```
from tensorflow.keras.models import load_model
model.save('/content/Digital Naturalist Dataset/Bird/Great Indian
Bustard Bird.h5')
```

```
from tensorflow.keras.models import load_model
import cv2
from skimage.transform import resize
model = load_model('/content/Digital Naturalist Dataset/Bird/Great
Indian Bustard Bird.h5')
def detect(frame):
  img = cv2.resize(frame,(224,224))
  img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
```

```
  if(np.max(img)>1):
    img = img/255.0
  img = np.array([img])
  prediction = model.predict(img)
  label = ["ABBOTTS BABBLER","ABBOTTS BOOBY","BALTIMORE
ORIOLE","BANANAQUIT","BAND TAILED GUAN","BLACKBURNIAM
WARBLER","CALIFORNIA GULL","CALIFORNIA QUAIL","CAMP"]
  preds = label[np.argmax(prediction)]
  return preds
import numpy as np
data = "/content/Digital Naturalist Dataset/Bird/Great Indian Bustard
Bird/download (1).jpg"
image = cv2.imread(data)
print(detect(image))
```

```
WARNING:tensorflow:6 out of the last 6 calls to <function
Model.make_predict_function.<locals>.predict_function at
0x7f7e4d3b35f0> triggered tf.function retracing. Tracing is expensive
and the excessive number of tracings could be due to (1) creating
@tf.function repeatedly in a loop, (2) passing tensors with different
shapes, (3) passing Python objects instead of tensors. For (1), please
define your @tf.function outside of the loop. For (2), @tf.function
has reduce_retracing=True option that can avoid unnecessary retracing.
For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more
details.

1/1 [==============================] - 1s 683ms/step
ABBOTTS BOOBY
```

For Flowers

MODEL BUILDING

1. Importing The Model Building Libraries

```
import tensorflow as tf
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import
ImageDataGenerator,load_img
from tensorflow.keras.models import Sequential
import numpy as np
from glob import glob
```

2. Loading The Model

```
IMAGE_SIZE = [224, 224]

train_path = '/content/Digital Naturalist Dataset/Flower'
valid_path = '/content/Digital Naturalist Dataset/Flower'
```

3. Adding Flatten Layer

```
vgg16 = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet',
include_top=False)
for layer in vgg16.layers:
    layer.trainable = False
folders = glob('/content/Digital Naturalist Dataset/Flower*')
folders
```

```
['/content/Digital Naturalist Dataset/Flower',
 '/content/Digital Naturalist Dataset/Flower.h5']
```

```
x = Flatten()(vgg16.output)
len(folders)
```

```
2
```

4. Adding Output Layer

```
prediction = Dense(len(folders), activation='softmax')(x)
```

5. Creating A Model Object

```
model = Model(inputs=vgg16.input, outputs=prediction)
model.summary()
```

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (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_3 (Flatten)         (None, 25088)             0

 dense_3 (Dense)             (None, 2)                 50178

=================================================================
Total params: 14,764,866
Trainable params: 50,178
Non-trainable params: 14,714,688
_________________________________________________________________
```

6. Configure The Learning Process

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

7. Train The Model

```
r = model.fit(
  xtrain2,
  validation_data=xtest2,
  epochs=10,
  steps_per_epoch=len(xtrain2),
  validation_steps=len(xtest2),
)
```

```
Epoch 1/10
5/5 [==============================] - 53s 12s/step - loss: 0.8370 -
```

```
accuracy: 0.5102 - val_loss: 0.1811 - val_accuracy: 0.9184
Epoch 2/10
5/5 [==============================] - 53s 12s/step - loss: 0.1051 -
accuracy: 0.9388 - val_loss: 0.1001 - val_accuracy: 0.9592
Epoch 3/10
5/5 [==============================] - 53s 12s/step - loss: 0.0755 -
accuracy: 0.9592 - val_loss: 0.0164 - val_accuracy: 1.0000
Epoch 4/10
5/5 [==============================] - 53s 12s/step - loss: 0.0066 -
accuracy: 1.0000 - val_loss: 0.0020 - val_accuracy: 1.0000
Epoch 5/10
5/5 [==============================] - 53s 12s/step - loss: 0.0040 -
accuracy: 1.0000 - val_loss: 0.0033 - val_accuracy: 1.0000
Epoch 6/10
5/5 [==============================] - 53s 12s/step - loss: 0.0068 -
accuracy: 1.0000 - val_loss: 0.0026 - val_accuracy: 1.0000
Epoch 7/10
5/5 [==============================] - 53s 12s/step - loss: 0.0027 -
accuracy: 1.0000 - val_loss: 0.0011 - val_accuracy: 1.0000
Epoch 8/10
5/5 [==============================] - 52s 12s/step - loss: 0.0033 -
accuracy: 1.0000 - val_loss: 5.3161e-04 - val_accuracy: 1.0000
Epoch 9/10
5/5 [==============================] - 52s 12s/step - loss: 5.7045e-04
- accuracy: 1.0000 - val_loss: 4.1401e-04 - val_accuracy: 1.0000
Epoch 10/10
5/5 [==============================] - 53s 12s/step - loss: 3.0292e-04
- accuracy: 1.0000 - val_loss: 4.3304e-04 - val_accuracy: 1.0000
```

8. Save And Test The Model

```
from tensorflow.keras.models import load_model
model.save('/content/Digital Naturalist Dataset/Flower.h5')
```

```
from tensorflow.keras.models import load_model
import cv2
from skimage.transform import resize
model = load_model('/content/Digital Naturalist Dataset/Flower.h5')
def detect(frame):
  img = cv2.resize(frame,(224,224))
  img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

  if(np.max(img)>1):
    img = img/255.0
  img = np.array([img])
  prediction = model.predict(img)
  label = ["daisy","dandelion","rose","sunflower","tulip"]
  preds = label[np.argmax(prediction)]
  return preds
import numpy as np
data = "/content/Digital Naturalist Dataset/Flower/Corpse
```

```
Flower/download (1).jpg"
image = cv2.imread(data)
print(detect(image))
```

```
1/1 [==============================] - 1s 667ms/step
daisy
```