FIRST_NAME = "Mel"

LAST_NAME = "Gerst"

STUDENT_ID = "800995291"

Final Class Project - ITCS 5154

Dog Breed Classifier - Student Mel Gerst

Duplicating project originally by TechVidvan

```
# Dog Breed Classifier
# ITCS 5154 - Student Mel Gerst
# Duplicating project by TechVidvan
# Import necessary packages for dog breed classifier
import cv2
import numpy as np
import pandas as pd
import tensorflow
import pathlib
import os

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import load_model,Model
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.layers import Dense,GlobalAveragePooling2D,Dropout,BatchNorma
from tensorflow.keras.applications.resnet_v2 import ResNet50V2,preprocess_input

print("Imports Complete")
print(pathlib.Path().resolve())

from google.colab import drive
drive.mount('/content/drive')
# Assuming your file is in "My Drive/data.csv"
# file_path = '/content/drive/My Drive/data.csv'
```

```
Imports Complete
/content
Drive already mounted at /content/drive; to attempt to forcibly remount, call
```

```
#specify number
num_breeds = 60
im_size = 224
batch_size = 64
encoder = LabelEncoder()
```

```
encoder = LabelEncoder()

#read the csv file
df_labels = pd.read_csv("/content/drive/My Drive/ColabNotebooks/DogBreedClassifie
#store training and testing images folder location
train_file = '/content/drive/My Drive/ColabNotebooks/DogBreedClassifierProject/AI
test_file =  '/content/drive/My Drive/ColabNotebooks/DogBreedClassifierProject/AI

#check the total number of unique breed in our dataset file
print("Total number of unique Dog Breeds in data:",len(df_labels.breed.unique()))
print(os.listdir(test_file))
```

```
Total number of unique Dog Breeds in data: 120
['e53cb5b42ea1a7700cd294a336890361.jpg', 'e7afcce6e45858fc3b294cc5c0b15a53.jpg
```

Start coding or generate with AI.

```
#get only 60 unique breeds record
breed_dict = list(df_labels['breed'].value_counts().keys())
new_list = sorted(breed_dict,reverse=True)[:num_breeds*2+1:2]
#change the dataset to have only those 60 unique breed records
df_labels = df_labels.query('breed in @new_list')
#create new column which will contain image name with the image extension
df_labels['img_file'] = df_labels['id'].apply(lambda x: x + ".jpg")
print("Total number of unique Dog Breeds used in model training:",len(df_labels.b
print("The breeds used for training and testing are:", sorted(df_labels.breed.uni
```

```
#create a numpy array of the shape
#(number of dataset records, image size , image size, 3 for rgb channel layer)
#this will be input for model
train_x = np.zeros((len(df_labels), im_size, im_size, 3), dtype='float32')
```

```
Total number of unique Dog Breeds used in model training: 60
The breeds used for training and testing are: ['afghan_hound', 'airedale', 'a
<ipython-input-3-343a299779ee>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
  df_labels['img_file'] = df_labels['id'].apply(lambda x: x + ".jpg")
```

```
#iterate over img_file column of our dataset
for i, img_id in enumerate(df_labels['img_file']):
  #read the image file and convert into numeric format
  #resize all images to one dimension i.e. 224x224
  #we will get array with the shape of
  #(224,224,3) where 3 is the RGB channels layers
  img = cv2.resize(cv2.imread(train_file+img_id,cv2.IMREAD_COLOR),((im_size,im_si
  #scale array into the range of -1 to 1.
  #preprocess the array and expand its dimension on the axis 0
  img_array = preprocess_input(np.expand_dims(np.array(img[...,::-1].astype(np.fl
  #update the train_x variable with new element
  train_x[i] = img_array
```

```
#this will be target for model.
#convert breed names into numerical format
train_y = encoder.fit_transform(df_labels["breed"].values)

#split the dataset in the ratio of 80:20.
#80% for training and 20% for testing purpose
x_train, x_test, y_train, y_test = train_test_split(train_x,train_y,test_size=0.2

#Image augmentation using ImageDataGenerator class
train_datagen = ImageDataGenerator(rotation_range=45,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.2,
                                   zoom_range=0.25,
                                   horizontal_flip=True
```

```
                                    horizontal_flip=True,
                                    fill_mode='nearest')

#generate images for training sets
train_generator = train_datagen.flow(x_train,
                                     y_train,
                                     batch_size=batch_size)

#same process for Testing sets also by declaring the instance
test_datagen = ImageDataGenerator()

test_generator = test_datagen.flow(x_test,
                                     y_test,
                                     batch_size=batch_size)
```

```
#building the model using ResNet50V2 with input shape of our image array
#weights for our network will be from of imagenet dataset
#we will not include the first Dense layer
resnet = ResNet50V2(input_shape = [im_size,im_size,3], weights='imagenet', includ
#freeze all trainable layers and train only top layers
for layer in resnet.layers:
    layer.trainable = False

#add global average pooling layer and Batch Normalization layer
x = resnet.output
x = BatchNormalization()(x)
x = GlobalAveragePooling2D()(x)
x = Dropout(0.5)(x)
#add fully connected layer
x = Dense(1024, activation='relu')(x)
x = Dropout(0.5)(x)

#add output layer having the shape equal to number of breeds
predictions = Dense(num_breeds, activation='softmax')(x)
```

```
#create model class with inputs and outputs
model = Model(inputs=resnet.input, outputs=predictions)
```

```
#model.summary()

#epochs for model training and learning rate for optimizer
epochs = 10
learning_rate = 1e-3

#using RMSprop optimizer compile or build the model
optimizer = RMSprop(learning_rate=learning_rate,rho=0.9)
model.compile(optimizer=optimizer,
              loss='sparse_categorical_crossentropy',
              metrics=["accuracy"])

#fit the training generator data and train the model
model.fit(train_generator,
                 steps_per_epoch= x_train.shape[0] // batch_size,
                 epochs= epochs,
                 validation_data= test_generator,
                 validation_steps= x_test.shape[0] // batch_size)

#Save the model for prediction
model.save("model.keras")

#load the model
model = load_model("model.keras")
```

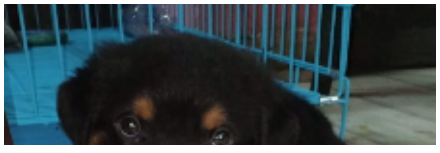
```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_da
  self._warn_if_super_not_called()
64/64 ━━━━━━━━━━━━━━━━━━━━ 1029s 16s/step - accuracy: 0.3116 - loss: 2.9106 -
Epoch 2/10
 1/64 ━━━━━━━━━━━━━━━━━━━━ 11:56 11s/step - accuracy: 0.6250 - loss: 1.4867/us
  self.gen.throw(typ, value, traceback)
64/64 ━━━━━━━━━━━━━━━━━━━━ 15s 54ms/step - accuracy: 0.6250 - loss: 1.4867 - v
Epoch 3/10
64/64 ━━━━━━━━━━━━━━━━━━━━ 1034s 16s/step - accuracy: 0.6197 - loss: 1.2991 -
Epoch 4/10
64/64 ━━━━━━━━━━━━━━━━━━━━ 14s 44ms/step - accuracy: 0.5938 - loss: 1.1805 - v
Epoch 5/10
64/64 ━━━━━━━━━━━━━━━━━━━━ 1022s 16s/step - accuracy: 0.6765 - loss: 1.0837 -
Epoch 6/10
64/64 ━━━━━━━━━━━━━━━━━━━━ 14s 26ms/step - accuracy: 0.6094 - loss: 1.0612 - v
Epoch 7/10
64/64 ━━━━━━━━━━━━━━━━━━━━ 1015s 16s/step - accuracy: 0.7038 - loss: 0.9997 -
Epoch 8/10
64/64 ━━━━━━━━━━━━━━━━━━━━ 14s 26ms/step - accuracy: 0.7031 - loss: 1.1159 - v
Epoch 9/10
64/64 ━━━━━━━━━━━━━━━━━━━━ 1020s 16s/step - accuracy: 0.6974 - loss: 0.9663 -
Epoch 10/10
64/64 ━━━━━━━━━━━━━━━━━━━━ 14s 26ms/step - accuracy: 0.7031 - loss: 0.7606 - v
```

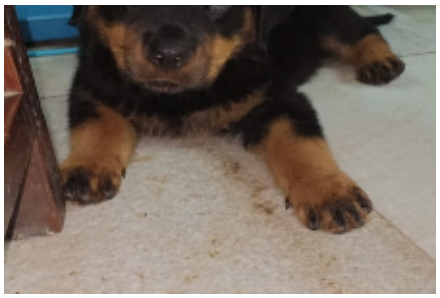
```
64/64 ━━━━━━━━━━━━━━━━━━━━ 14s 208ms/step - accuracy: 0.7051 - loss: 0.7000 - v
```

```
#get the image of the dog #1 for prediction
pred_img_path = '/content/drive/My Drive/ColabNotebooks/DogBreedClassifierProject/A
#read the image file and convert into numeric format
#resize all images to one dimension i.e. 224x224
pred_img_array = cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((im_size,im
#scale array into the range of -1 to 1.
#expand the dimesion on the axis 0 and normalize the array values
pred_img_array = preprocess_input(np.expand_dims(np.array(pred_img_array[...,::-1].

#feed the model with the image array for prediction
pred_val = model.predict(np.array(pred_img_array,dtype="float32"))
#display the image of dog
from google.colab.patches import cv2_imshow
cv2_imshow(cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((im_size,im_size)
# cv2.imshow("TechVidvan",cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((i
#display the predicted breed of dog
pred_breed = sorted(new_list)[np.argmax(pred_val)]
print("Predicted Breed for this Dog is :",pred_breed)

#get the image of the dog #2 for prediction
pred_img_path2 = '/content/drive/My Drive/ColabNotebooks/DogBreedClassifierProject/
#read the image file and convert into numeric format
#resize all images to one dimension i.e. 224x224
pred_img_array2 = cv2.resize(cv2.imread(pred_img_path2,cv2.IMREAD_COLOR),((im_size,
#scale array into the range of -1 to 1.
#expand the dimesion on the axis 0 and normalize the array values
pred_img_array2 = preprocess_input(np.expand_dims(np.array(pred_img_array2[...,::-1

#feed the model with the image array for prediction
pred_val2 = model.predict(np.array(pred_img_array2,dtype="float32"))
#display the image of dog
from google.colab.patches import cv2_imshow
cv2_imshow(cv2.resize(cv2.imread(pred_img_path2,cv2.IMREAD_COLOR),((im_size,im_size
# cv2.imshow("TechVidvan",cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((i
#display the predicted breed of dog
pred_breed2 = sorted(new_list)[np.argmax(pred_val2)]
print("Predicted Breed for this Dog is :",pred_breed2)

#get the image of the dog #3 for prediction
pred_img_path3 = '/content/drive/My Drive/ColabNotebooks/DogBreedClassifierProject/
#read the image file and convert into numeric format
#resize all images to one dimension i.e. 224x224
pred_img_array3 = cv2.resize(cv2.imread(pred_img_path3,cv2.IMREAD_COLOR),((im_size,
#scale array into the range of -1 to 1.
#expand the dimesion on the axis 0 and normalize the array values
pred_img_array3 = preprocess_input(np.expand_dims(np.array(pred_img_array3[...,::-1

#feed the model with the image array for prediction
```

```
pred_val3 = model.predict(np.array(pred_img_array3,dtype="float32"))
#display the image of dog
from google.colab.patches import cv2_imshow
cv2_imshow(cv2.resize(cv2.imread(pred_img_path3,cv2.IMREAD_COLOR),((im_size,im_size
# cv2.imshow("TechVidvan",cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((i
#display the predicted breed of dog
pred_breed3 = sorted(new_list)[np.argmax(pred_val3)]
print("Predicted Breed for this Dog is :",pred_breed3)

#get the image of the dog #4 for prediction
pred_img_path4 = '/content/drive/My Drive/ColabNotebooks/DogBreedClassifierProject/
#read the image file and convert into numeric format
#resize all images to one dimension i.e. 224x224
pred_img_array4 = cv2.resize(cv2.imread(pred_img_path4,cv2.IMREAD_COLOR),((im_size,
#scale array into the range of -1 to 1.
#expand the dimesion on the axis 0 and normalize the array values
pred_img_array4 = preprocess_input(np.expand_dims(np.array(pred_img_array4[...,::-1

#feed the model with the image array for prediction
pred_val4 = model.predict(np.array(pred_img_array4,dtype="float32"))
#display the image of dog
from google.colab.patches import cv2_imshow
cv2_imshow(cv2.resize(cv2.imread(pred_img_path4,cv2.IMREAD_COLOR),((im_size,im_size
# cv2.imshow("TechVidvan",cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((i
#display the predicted breed of dog
pred_breed4 = sorted(new_list)[np.argmax(pred_val4)]
print("Predicted Breed for this Dog is :",pred_breed4)

#get the image of the dog #5 for prediction
pred_img_path5 = '/content/drive/My Drive/ColabNotebooks/DogBreedClassifierProject/
#read the image file and convert into numeric format
#resize all images to one dimension i.e. 224x224
pred_img_array5 = cv2.resize(cv2.imread(pred_img_path5,cv2.IMREAD_COLOR),((im_size,
#scale array into the range of -1 to 1.
#expand the dimesion on the axis 0 and normalize the array values
pred_img_array5 = preprocess_input(np.expand_dims(np.array(pred_img_array5[...,::-1

#feed the model with the image array for prediction
pred_val5 = model.predict(np.array(pred_img_array5,dtype="float32"))
#display the image of dog
from google.colab.patches import cv2_imshow
cv2_imshow(cv2.resize(cv2.imread(pred_img_path5,cv2.IMREAD_COLOR),((im_size,im_size
#display the predicted breed of dog
pred_breed5 = sorted(new_list)[np.argmax(pred_val5)]
print("Predicted Breed for this Dog is :",pred_breed5)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 198ms/step
```





```
Predicted Breed for this Dog is : rottweiler
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 206ms/step
```

```
Predicted Breed for this Dog is : miniature_pinscher
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 190ms/step
```

```
Predicted Breed for this Dog is : labrador_retriever
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 177ms/step
```

```
Predicted Breed for this Dog is : scottish_deerhound
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 321ms/step
```

```
Predicted Breed for this Dog is : whippet
```