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,BatchNor
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')
# My files are mounted in Google drive for access by Colab, stored in "My Drive/d
# file_path = '/content/drive/My Drive/data.csv'
```

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

```
# Initialize Variables
encoder = LabelEncoder()
image_size = 224
breed_count = 60
batch_size = 64



# Grab input files and data
df_labels = pd.read_csv("/content/drive/My Drive/ColabNotebooks/DogBreedClassifie
#store training and testing images folder location
training_data = '/content/drive/My Drive/ColabNotebooks/DogBreedClassifierProject
testing_data =  '/content/drive/My Drive/ColabNotebooks/DogBreedClassifierProject

# Check and print the total number of unique breeds in original dataset
print("Total number of unique Dog Breeds in data:",len(df_labels.breed.unique()))
print(os.listdir(testing_data))
```

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

```
Start coding or generate with AI.
```

```
# Drop breeds considered to 60 breeds to speed up runtimes
breed_dict = list(df_labels['breed'].value_counts().keys())
new_list = sorted(breed_dict,reverse=True)[:breed_count*2:2]
# Limit dataset to have only those 60 unique breed records
df_labels = df_labels.query('breed in @new_list')
# Add 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 , imag
# Input for model
train_x = np.zeros((len(df_labels), image_size, image_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', 'ap
<ipython-input-3-e80864a923a5>: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 set by image size
  # We will get array with the shape of
  # (224,224,3) where 3 is the RGB channels layers
  img = cv2.resize(cv2.imread(training_data+img_id,cv2.IMREAD_COLOR),((image_size
  # 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,
                                   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)
```

```
# Build 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 = [image_size,image_size,3], weights='imagenet',
# 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(breed_count, activation='softmax')(x)

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

```
# model.summary()

# Set the num_epochs for model training and learning rate for optimizer
num_epochs = 20
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= num_epochs,
                 validation_data= test_generator,
                 validation_steps= x_test.shape[0] // batch_size)

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

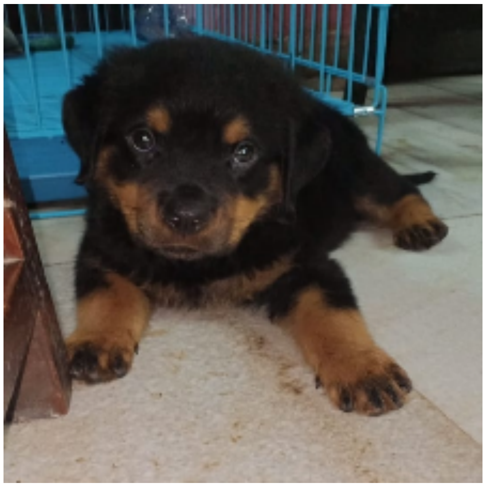
```
# Load the model
model = load_model("model.keras")
```

```
Epoch 1/20
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_da
  self._warn_if_super_not_called()
64/64 ━━━━━━━━━━━━━━━━━━━━ 1023s 16s/step - accuracy: 0.2902 - loss: 2.9611 -
Epoch 2/20
 1/64 ━━━━━━━━━━━━━━━━━━━━ 11:35 11s/step - accuracy: 0.6250 - loss: 1.2642/us
  self.gen.throw(typ, value, traceback)
64/64 ━━━━━━━━━━━━━━━━━━━━ 70s 939ms/step - accuracy: 0.6250 - loss: 1.2642 -
Epoch 3/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 1049s 16s/step - accuracy: 0.6114 - loss: 1.2926 -
Epoch 4/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 31s 294ms/step - accuracy: 0.7188 - loss: 0.9045 -
Epoch 5/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 997s 16s/step - accuracy: 0.6773 - loss: 1.0788 - v
Epoch 6/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 14s 26ms/step - accuracy: 0.6562 - loss: 1.1297 - v
Epoch 7/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 1034s 16s/step - accuracy: 0.6972 - loss: 1.0015 -
Epoch 8/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 14s 25ms/step - accuracy: 0.7344 - loss: 0.9811 - v
Epoch 9/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 1033s 16s/step - accuracy: 0.7200 - loss: 0.9321 -
Epoch 10/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 14s 26ms/step - accuracy: 0.5938 - loss: 1.1528 - v
Epoch 11/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 1027s 16s/step - accuracy: 0.7262 - loss: 0.8610 -
Epoch 12/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 14s 46ms/step - accuracy: 0.6875 - loss: 1.1535 - v
Epoch 13/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 999s 16s/step - accuracy: 0.7386 - loss: 0.8100 - v
Epoch 14/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 14s 26ms/step - accuracy: 0.7188 - loss: 0.9247 - v
Epoch 15/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 1000s 16s/step - accuracy: 0.7351 - loss: 0.8603 -
Epoch 16/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 14s 26ms/step - accuracy: 0.7188 - loss: 0.9018 - v
Epoch 17/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 998s 16s/step - accuracy: 0.7568 - loss: 0.8031 - v
Epoch 18/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 14s 25ms/step - accuracy: 0.7188 - loss: 0.9588 - v
Epoch 19/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 1022s 15s/step - accuracy: 0.7740 - loss: 0.7466 -
Epoch 20/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 14s 26ms/step - accuracy: 0.7969 - loss: 0.8753 - v
```

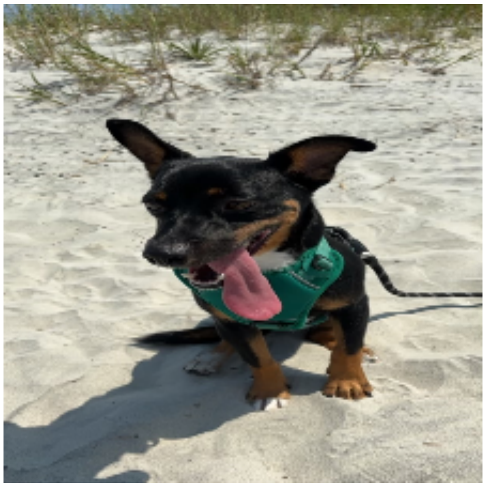
```
# Get the image of the dog #1 for prediction
pred_img_path = '/content/drive/My Drive/ColabNotebooks/DogBreedClassifierProject/A
# Read the image file and convert into numeric format
# Resize all images to one dimension i.e. 224x224
pred_img_array = cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((image_size
# 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),((image_size,image
# cv2.imshow("TechVidvan",cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((i
# Display the predicted breed of dog
predicted_breed = sorted(new_list)[np.argmax(pred_val)]
print("Predicted Breed for this Dog is :",predicted_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),((image_si
# 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),((image_size,imag
# cv2.imshow("TechVidvan",cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((i
# Display the predicted breed of dog
predicted_breed2 = sorted(new_list)[np.argmax(pred_val2)]
print("Predicted Breed for this Dog is :",predicted_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),((image_si
# 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),((image_size,imag
# cv2.imshow("TechVidvan",cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((i
# Display the predicted breed of dog
predicted_breed3 = sorted(new_list)[np.argmax(pred_val3)]
print("Predicted Breed for this Dog is :",predicted_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),((image_si
# 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),((image_size,imag
# cv2.imshow("TechVidvan",cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((i
# Display the predicted breed of dog
predicted_breed4 = sorted(new_list)[np.argmax(pred_val4)]
print("Predicted Breed for this Dog is :",predicted_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),((image_si
# 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),((image_size,imag
#Display the predicted breed of dog
predicted_breed5 = sorted(new_list)[np.argmax(pred_val5)]
print("Predicted Breed for this Dog is :",predicted_breed5)
```

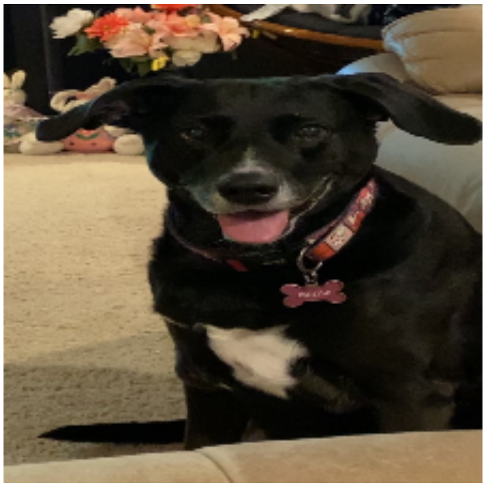
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
```



```
Predicted Breed for this Dog is : rottweiler
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 193ms/step
```

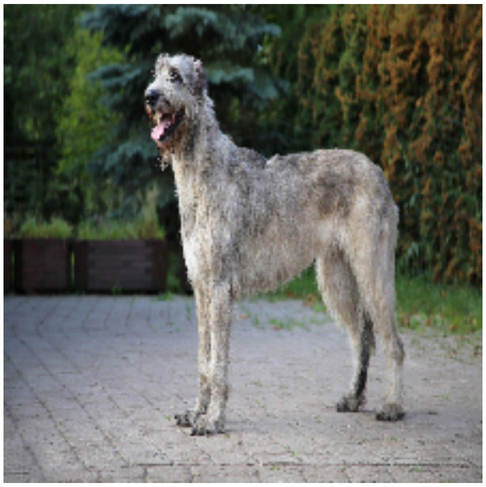


```
Predicted Breed for this Dog is : miniature_pinscher
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 190ms/step
```

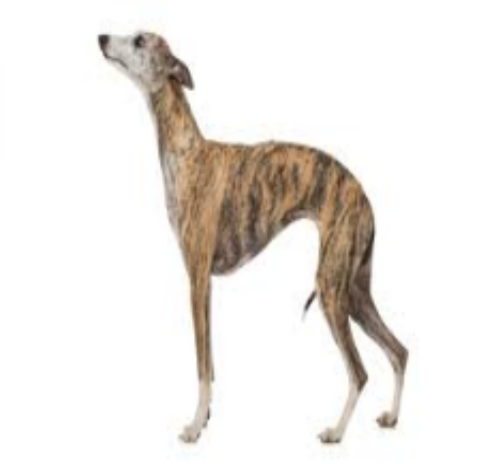


```
Predicted Breed for this Dog is : labrador_retriever
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 271ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 171ms/step
```



```
Predicted Breed for this Dog is : scottish_deerhound
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 323ms/step
```



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