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

Resources

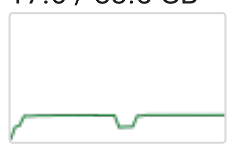
You are not subscribed. Learn more
Available: 50.66 compute units
Usage rate: approximately 10.59 per hour
You have 1 active session.

Manage sessions

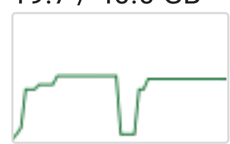
Want more memory and disk space?
Upgrade to Colab Pro

Python 3 Google Compute Engine backend (GPU)
Showing resources from 4:40 PM to 6:15 PM

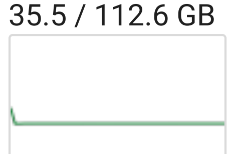
System RAM
17.6 / 83.5 GB



GPU RAM
19.7 / 40.0 GB



Disk
35.5 / 112.6 GB



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

from tensorflow.keras.preprocessing.image
from sklearn.model_selection import train_
from sklearn.preprocessing import LabelEnc
from tensorflow.keras.models import load_m
from tensorflow.keras.optimizers import RM
from tensorflow.keras.layers import Dense,
from tensorflow.keras.applications.resnet_

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

from google.colab import drive
# drive._mount('/content/drive')
drive.mount('/content/drive/')
# My files are mounted in Google drive for
# file_path = '/content/drive/My Drive/dat
```

```
Imports Complete
/content
Drive already mounted at /content/driv
```

```
# 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
#store training and testing images folder
training_data = '/content/drive/My Drive/(
testing_data =  '/content/drive/My Drive/(

# Check and print the total number of unic
print("Total number of unique Dog Breeds i
print(os.listdir(testing_data))
```

```
Total number of unique Dog Breeds in d
['06b727fc8e24e46fd7ea78b08091cab5.jpg
```

```
# Drop breeds considered to 60 breeds to s
breed_dict = list(df_labels['breed'].value
new_list = sorted(breed_dict,reverse=True)
# Limit dataset to have only those 60 unic
df_labels = df_labels.query('breed in @new
# Add new column which will contain image
df_labels['img_file'] = df_labels['id'].ap
print("Total number of unique Dog Breeds u
print("The breeds used for training and te

# Create a numpy array of the shape (numbe
# Input for model
train_x = np.zeros((len(df_labels), image_
```

```
Total number of unique Dog Breeds used
The breeds used for training and testi
<ipython-input-3-e80864a923a5>:7: Sett
A value is trying to be set on a copy
Try using .loc[row_indexer,col_indexer

See the caveats in the documentation:
  df_labels['img_file'] = df_labels['i
```

```
# #iterate over img_file column of our dat
# for i, img_id in enumerate(df_labels['im
#   # Read the image file and convert into
#   # Resize all images to one dimension i
#   # We will get array with the shape of
#   # (224,224,3) where 3 is the RGB chann
#   img = cv2.resize(cv2.imread(training_c
#   # Scale array into the range of -1 to
#   # Preprocess the array and expand its
#   img_array = preprocess_input(np.expanc
#   # Update the train_x variable with new
#   train_x[i] = img_array
```

```
# print(train_x.shape)
# np.save('/content/drive/My Drive/ColabNc
```

```
train_x = np.load('/content/drive/My Drive
print(train_x.shape)
```

```
(5175, 224, 224, 3)
```

```
# This will be target for model.
# Convert breed names into numerical forma
train_y = encoder.fit_transform(df_labels[

# Split the dataset in the ratio of 80:20.
#80% for training and 20% for testing purp
x_train, x_test, y_train, y_test = train_t
```

```
#Image augmentation using ImageDataGenerat
train_datagen = ImageDataGenerator(rotatio
                                   width_s
                                   height_
                                   shear_r
                                   zoom_ra
                                   horizon
                                   fill_mo

# Generate images for training sets
train_generator = train_datagen.flow(x_tra
                                     y_tra
                                     batch

# Same process for Testing sets also by de
test_datagen = ImageDataGenerator()

test_generator = test_datagen.flow(x_test,
                                     y_tes
                                     batch
```

```
# Model #1 - Build the model using ResNet5
# Weights for our network will be from of
# We will not include the first Dense laye
resnet = ResNet50V2(input_shape = [image_s
# Freeze all trainable layers and train or
for layer in resnet.layers:
    layer.trainable = False

# Add global average pooling layer and Bat
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
predictions = Dense(breed_count, activatio

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

```
# model.summary()

# Set the num_epochs for model training ar
num_epochs = 20
learning_rate = 1e-3

# Using RMSprop optimizer compile or builc
optimizer = RMSprop(learning_rate=learning
model.compile(optimizer=optimizer,
              loss='sparse_categorical_crc
              metrics=["accuracy"])

# Fit the training generator data and trai
model.fit(train_generator,
                 steps_per_epoch= x_train.
                 epochs= num_epochs,
                 validation_data= test_ger
                 validation_steps= x_test.

test_loss, test_accuracy = model.evaluate(
```

```
print(f'Resnet Test accuracy: {test_accura

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

```
Epoch 1/20
/usr/local/lib/python3.10/dist-package
  self._warn_if_super_not_called()
64/64 ━━━━━━━━━━━━━━━━━━━━ 69s 789ms/s
Epoch 2/20
 1/64 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/ste
  self.gen.throw(typ, value, traceback
64/64 ━━━━━━━━━━━━━━━━━━━━ 5s 83ms/ste
Epoch 3/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 42s 583ms/s
Epoch 4/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 405us/st
Epoch 5/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 42s 585ms/s
Epoch 6/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 408us/st
Epoch 7/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 42s 583ms/s
Epoch 8/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 394us/st
Epoch 9/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 42s 583ms/s
Epoch 10/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 400us/st
Epoch 11/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 42s 582ms/s
Epoch 12/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 417us/st
Epoch 13/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 42s 580ms/s
Epoch 14/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 400us/st
Epoch 15/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 42s 581ms/s
Epoch 16/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 404us/st
Epoch 17/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 42s 581ms/s
Epoch 18/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 400us/st
Epoch 19/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 42s 581ms/s
Epoch 20/20
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 397us/st
33/33 ━━━━━━━━━━━━━━━━━━━━ 7s 15ms/ste
Resnet Test accuracy: 0.78
```

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

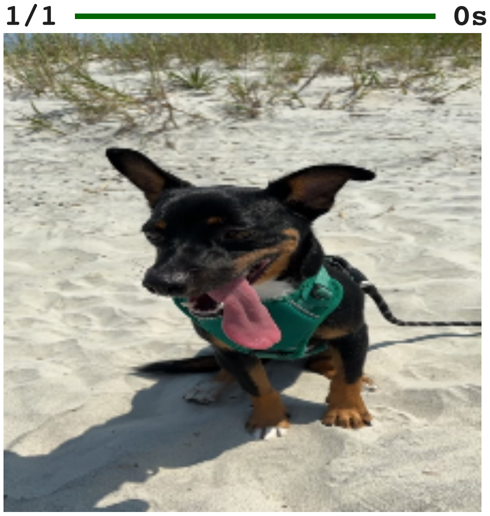
```
# Get the image of the dog #1 for predicti
pred_img_path = '/content/drive/My Drive/(
# Read the image file and convert into num
# Resize all images to one dimension i.e.
pred_img_array = cv2.resize(cv2.imread(pre
# Scale array into the range of -1 to 1.
# Expand the dimesion on the axis 0 and nc
pred_img_array = preprocess_input(np.expar

# Feed the model with the image array for
pred_val = model.predict(np.array(pred_img
# Display the image of dog
from google.colab.patches import cv2_imshc
cv2_imshow(cv2.resize(cv2.imread(pred_img_
# Display the predicted breed of dog
predicted_breed = sorted(new_list)[np.argm
print("Predicted Breed for this Dog is :",

# Get the image of the dog #2 for predicti
pred_img_path2 = '/content/drive/My Drive/
# Read the image file and convert into num
# Resize all images to one dimension i.e.
pred_img_array2 = cv2.resize(cv2.imread(pr
# Scale array into the range of -1 to 1.
# Expand the dimesion on the axis 0 and nc
pred_img_array2 = preprocess_input(np.expa

# Feed the model with the image array for
pred_val2 = model.predict(np.array(pred_im
# Display the image of dog
from google.colab.patches import cv2_imshc
cv2_imshow(cv2.resize(cv2.imread(pred_img_
# Display the predicted breed of dog
predicted_breed2 = sorted(new_list)[np.arg
print("Predicted Breed for this Dog is :",

# Get the image of the dog #3 for predicti
pred_img_path3 = '/content/drive/My Drive/
# Read the image file and convert into num
# Resize all images to one dimension i.e.
```

```
pred_img_array3 = cv2.resize(cv2.imread(pr
# Scale array into the range of -1 to 1.
# Expand the dimesion on the axis 0 and nc
pred_img_array3 = preprocess_input(np.expa

# Feed the model with the image array for
pred_val3 = model.predict(np.array(pred_im
# Display the image of dog
from google.colab.patches import cv2_imshc
cv2_imshow(cv2.resize(cv2.imread(pred_img_
# Display the predicted breed of dog
predicted_breed3 = sorted(new_list)[np.arg
print("Predicted Breed for this Dog is :",

# Get the image of the dog #4 for predicti
pred_img_path4 = '/content/drive/My Drive/
# Read the image file and convert into num
# Resize all images to one dimension i.e.
pred_img_array4 = cv2.resize(cv2.imread(pr
# Scale array into the range of -1 to 1.
# Expand the dimesion on the axis 0 and nc
pred_img_array4 = preprocess_input(np.expa

# Feed the model with the image array for
pred_val4 = model.predict(np.array(pred_im
# Display the image of dog
from google.colab.patches import cv2_imshc
cv2_imshow(cv2.resize(cv2.imread(pred_img_
# Display the predicted breed of dog
predicted_breed4 = sorted(new_list)[np.arg
print("Predicted Breed for this Dog is :",

# Get the image of the dog #5 for predicti
pred_img_path5 = '/content/drive/My Drive/
# Read the image file and convert into num
# Resize all images to one dimension i.e.
pred_img_array5 = cv2.resize(cv2.imread(pr
# Scale array into the range of -1 to 1.
# Expand the dimesion on the axis 0 and nc
pred_img_array5 = preprocess_input(np.expa

# Feed the model with the image array for
pred_val5 = model.predict(np.array(pred_im
# Display the image of dog
from google.colab.patches import cv2_imshc
```

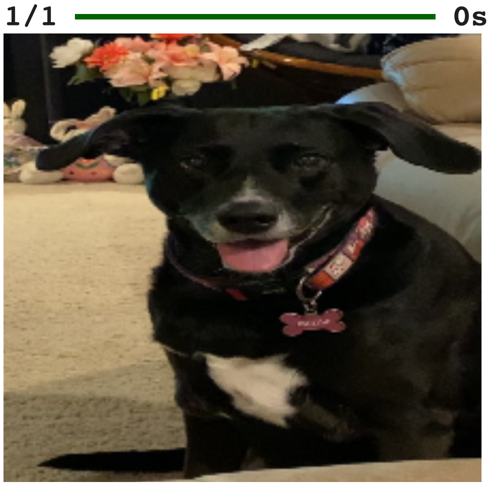
```
cv2_imshow(cv2.resize(cv2.imread(pred_img_
#Display the predicted breed of dog
predicted_breed5 = sorted(new_list)[np.arg
print("Predicted Breed for this Dog is :",
print("Check image size: ", image_size)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 5s 5s/step
```

```
Predicted Breed for this Dog is : rott
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
```



```
Predicted Breed for this Dog is : mini
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
```



```
Predicted Breed for this Dog is : labr
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
```

```
Predicted Breed for this Dog is : scot
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
```

```
Predicted Breed for this Dog is : whip
Check image size:  224
```

```
#VGG model
import tensorflow as tf
from tensorflow.keras import layers, model
from sklearn.model_selection import train_

# This will be target for model.
# Convert breed names into numerical forma
train_y = encoder.fit_transform(df_labels[

# Assuming train_y is one-hot encoded, if
train_y = tf.keras.utils.to_categorical(tr

# Split the data
```

```
x_train, x_test, y_train, y_test = train_t


def create_vgg_model(input_shape, num_clas
    base_model = tf.keras.applications.VGG

    # Freeze the base model
    base_model.trainable = False

    model = models.Sequential([
        base_model,
        layers.Flatten(),
        layers.Dense(512, activation='relu
        layers.Dropout(0.5),
        layers.Dense(num_classes, activati
    ])

    return model, base_model # Return both

input_shape = (224, 224, 3)
num_classes = 60
model, base_model = create_vgg_model(input

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

history = model.fit(x_train, y_train,
                    epochs=50,
                    batch_size=32,
                    validation_data=(x_tes

test_loss, test_accuracy = model.evaluate(
print(f'Test accuracy: {test_accuracy:.2f}

base_model.trainable = True  # Now base_mo
for layer in base_model.layers[:-4]:  # Un
    layer.trainable = False

model.compile(optimizer=tf.keras.optimizer
              loss='categorical_crossentro
              metrics=['accuracy'])

history_finetune = model.fit(x_train, y_tr
                             epochs=10,
```

```
                                        batch_size=3
                                        validation_c

test_loss, test_accuracy = model.evaluate(
print(f'VGG Test accuracy: {test_accuracy:
```

```
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/s
Epoch 33/50
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/s
Epoch 34/50
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/s
Epoch 35/50
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/s
Epoch 36/50
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/s
Epoch 37/50
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/s
Epoch 38/50
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/s
Epoch 39/50
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/s
Epoch 40/50
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/s
Epoch 41/50
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/s
Epoch 42/50
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/s
Epoch 43/50
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/s
Epoch 44/50
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/s
Epoch 45/50
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/s
Epoch 46/50
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/s
Epoch 47/50
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/s
Epoch 48/50
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/s
Epoch 49/50
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/s
Epoch 50/50
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/s
33/33 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/ste
Test accuracy: 0.34
Epoch 1/10
130/130 ━━━━━━━━━━━━━━━━━━━━ 11s 59ms/
Epoch 2/10
```

```
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/s
Epoch 3/10
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/s
Epoch 4/10
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/s
Epoch 5/10
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/s
Epoch 6/10
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/s
Epoch 7/10
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/s
Epoch 8/10
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/s
Epoch 9/10
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/s
Epoch 10/10
130/130 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/s
33/33 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/ste
```

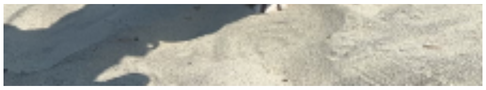
```
# Second round of predictions with VGG moc
print("Check image size: ", image_size)
# Get the image of the dog #1 for predicti
pred_img_path = '/content/drive/My Drive/C
# Read the image file and convert into num
# Resize all images to one dimension i.e.
pred_img_array = cv2.resize(cv2.imread(pre
# Scale array into the range of -1 to 1.
# Expand the dimesion on the axis 0 and nc
pred_img_array = preprocess_input(np.expar

# Feed the model with the image array for
pred_val = model.predict(np.array(pred_img
# Display the image of dog
from google.colab.patches import cv2_imshc
cv2_imshow(cv2.resize(cv2.imread(pred_img_
# Display the predicted breed of dog
predicted_breed = sorted(new_list)[np.argm
print("Predicted Breed for this Dog is :",

# Get the image of the dog #2 for predicti
pred_img_path2 = '/content/drive/My Drive/
# Read the image file and convert into num
# Resize all images to one dimension i.e.
pred_img_array2 = cv2.resize(cv2.imread(pr
# Scale array into the range of -1 to 1.
# Expand the dimesion on the axis 0 and nc
```

```
pred_img_array2 = preprocess_input(np.expa

# Feed the model with the image array for
pred_val2 = model.predict(np.array(pred_im
# Display the image of dog
from google.colab.patches import cv2_imshc
cv2_imshow(cv2.resize(cv2.imread(pred_img_
# Display the predicted breed of dog
predicted_breed2 = sorted(new_list)[np.arg
print("Predicted Breed for this Dog is :",

# Get the image of the dog #3 for predicti
pred_img_path3 = '/content/drive/My Drive/
# Read the image file and convert into num
# Resize all images to one dimension i.e.
pred_img_array3 = cv2.resize(cv2.imread(pr
# Scale array into the range of -1 to 1.
# Expand the dimesion on the axis 0 and nc
pred_img_array3 = preprocess_input(np.expa

# Feed the model with the image array for
pred_val3 = model.predict(np.array(pred_im
# Display the image of dog
from google.colab.patches import cv2_imshc
cv2_imshow(cv2.resize(cv2.imread(pred_img_
# Display the predicted breed of dog
predicted_breed3 = sorted(new_list)[np.arg
print("Predicted Breed for this Dog is :",

# Get the image of the dog #4 for predicti
pred_img_path4 = '/content/drive/My Drive/
# Read the image file and convert into num
# Resize all images to one dimension i.e.
pred_img_array4 = cv2.resize(cv2.imread(pr
# Scale array into the range of -1 to 1.
# Expand the dimesion on the axis 0 and nc
pred_img_array4 = preprocess_input(np.expa

# Feed the model with the image array for
pred_val4 = model.predict(np.array(pred_im
# Display the image of dog
from google.colab.patches import cv2_imshc
cv2_imshow(cv2.resize(cv2.imread(pred_img_
# Display the predicted breed of dog
predicted_breed4 = sorted(new_list)[np.arg
```

```
print("Predicted Breed for this Dog is :",

# Get the image of the dog #5 for predicti
pred_img_path5 = '/content/drive/My Drive/
# Read the image file and convert into nun
# Resize all images to one dimension i.e.
pred_img_array5 = cv2.resize(cv2.imread(pr
# Scale array into the range of -1 to 1.
# Expand the dimesion on the axis 0 and nc
pred_img_array5 = preprocess_input(np.expa

# Feed the model with the image array for
pred_val5 = model.predict(np.array(pred_in
# Display the image of dog
from google.colab.patches import cv2_imshc
cv2_imshow(cv2.resize(cv2.imread(pred_img_
#Display the predicted breed of dog
predicted_breed5 = sorted(new_list)[np.arg
print("Predicted Breed for this Dog is :",
```

```
Check image size:  224
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
```

```
Predicted Breed for this Dog is : rott
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```



```
Predicted Breed for this Dog is : mini
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
```

```
Predicted Breed for this Dog is : boxe
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```

```
Predicted Breed for this Dog is : iris
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
```

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

```
print(len(np.unique(y_train)))
```

2

```
# new SimpleCNN model
import torch
import torch.nn as nn
import torchvision.transforms as transforms
from torch.utils.data import DataLoader, Da
from sklearn.model_selection import train_t
import numpy as np

# Assume train_x and train_y are defined
# train_x: numpy array of shape (5175, 224,
# train_y: numpy array of shape (5175,)

# Split the dataset
# x_train, x_test, y_train, y_test = train_

# Set batch size
# batch_size = 64

# Define data augmentation and normalizatio
train_transforms = transforms.Compose([
    transforms.ToPILImage(),  # Convert num
    transforms.RandomRotation(45),
    transforms.RandomAffine(degrees=0, tran
    transforms.RandomResizedCrop(size=(224,
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
])

test_transforms = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])

# Custom Dataset Class
class CustomDataset(Dataset):
    def __init__(self, images, labels, tran
        self.images = images
        self.labels = labels
        self.transform = transform
```

```
    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        image = self.images[idx]
        label = self.labels[idx]

        if self.transform:
            image = self.transform(image)

        return image, label

# Create datasets
y_train_indices = np.argmax(y_train, axis=1
y_test_indices = np.argmax(y_test, axis=1)
train_dataset = CustomDataset(x_train, y_tr
test_dataset = CustomDataset(x_test, y_test

# Create DataLoaders
train_loader = DataLoader(train_dataset, ba
test_loader = DataLoader(test_dataset, batc

# Define the SimpleCNN model
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kerne
        self.pool = nn.MaxPool2d(kernel_siz
        self.conv2 = nn.Conv2d(32, 64, kern
        self.conv3 = nn.Conv2d(64, 128, ker

        # Calculate the input size for the
        self.fc1_input_size = 128 * (224 //
        self.fc1 = nn.Linear(self.fc1_input
        self.fc2 = nn.Linear(256, 60)  # Ou

    def forward(self, x):
        x = self.conv1(x)
        x = nn.ReLU()(x)
        x = self.pool(x)

        x = self.conv2(x)
        x = nn.ReLU()(x)
        x = self.pool(x)
```

```
        x = self.conv3(x)
        x = nn.ReLU()(x)
        x = self.pool(x)

        x = torch.flatten(x, 1)  # Flatten
        x = self.fc1(x)
        x = nn.ReLU()(x)
        x = self.fc2(x)
        return nn.LogSoftmax(dim=1)(x)

# Initialize model, loss function, and opti
device = torch.device("cuda" if torch.cuda.
model = SimpleCNN().to(device)
criterion = nn.NLLLoss()
optimizer = torch.optim.Adam(model.paramete

# Training Loop
num_epochs = 200
for epoch in range(num_epochs):
    model.train()  # Set model to training
    for images, labels in train_loader:
        images, labels = images.to(device),

        optimizer.zero_grad()  # Clear grad
        outputs = model(images)  # Forward
        loss = criterion(outputs, labels)
        loss.backward()  # Backward pass
        optimizer.step()  # Update weights

    print(f'Epoch [{epoch + 1}/{num_epochs}

# Evaluation on the test set
model.eval()  # Set model to evaluation mod
test_loss = 0
correct = 0

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device),
        outputs = model(images)
        test_loss += criterion(outputs, lab
        pred = outputs.argmax(dim=1)  # Get
        correct += (pred == labels).sum().i
```

```
test_accuracy = correct / len(test_dataset)
print(f'Test Loss: {test_loss/len(test_load
```

```
Epoch [142/200], Loss: 2.4036
Epoch [143/200], Loss: 2.2013
Epoch [144/200], Loss: 2.0762
Epoch [145/200], Loss: 2.0533
Epoch [146/200], Loss: 1.9337
Epoch [147/200], Loss: 2.3631
Epoch [148/200], Loss: 2.1085
Epoch [149/200], Loss: 1.9786
Epoch [150/200], Loss: 2.1217
Epoch [151/200], Loss: 2.1622
Epoch [152/200], Loss: 2.2347
Epoch [153/200], Loss: 2.1163
Epoch [154/200], Loss: 2.4530
Epoch [155/200], Loss: 2.5614
Epoch [156/200], Loss: 2.2602
Epoch [157/200], Loss: 1.7063
Epoch [158/200], Loss: 1.9796
Epoch [159/200], Loss: 1.9819
Epoch [160/200], Loss: 2.4385
Epoch [161/200], Loss: 1.5695
Epoch [162/200], Loss: 1.6115
Epoch [163/200], Loss: 1.6151
Epoch [164/200], Loss: 2.1459
Epoch [165/200], Loss: 2.3970
Epoch [166/200], Loss: 2.2453
Epoch [167/200], Loss: 1.8896
Epoch [168/200], Loss: 1.8341
Epoch [169/200], Loss: 2.0265
Epoch [170/200], Loss: 2.2931
Epoch [171/200], Loss: 1.7738
Epoch [172/200], Loss: 1.8996
Epoch [173/200], Loss: 2.0559
Epoch [174/200], Loss: 2.3785
Epoch [175/200], Loss: 1.7810
Epoch [176/200], Loss: 1.9395
Epoch [177/200], Loss: 2.1476
Epoch [178/200], Loss: 1.5677
Epoch [179/200], Loss: 1.9600
Epoch [180/200], Loss: 1.8884
Epoch [181/200], Loss: 2.0575
Epoch [182/200], Loss: 1.3027
Epoch [183/200], Loss: 2.1412
Epoch [184/200], Loss: 1.7419
Epoch [185/200], Loss: 1.8414
Epoch [186/200], Loss: 2.4970
Epoch [187/200], Loss: 1.4430
```

```
Epoch [188/200], Loss: 1.9959
Epoch [189/200], Loss: 1.5419
Epoch [190/200], Loss: 2.1794
Epoch [191/200], Loss: 1.6998
Epoch [192/200], Loss: 1.3166
Epoch [193/200], Loss: 1.8938
Epoch [194/200], Loss: 1.6965
Epoch [195/200], Loss: 2.0326
Epoch [196/200], Loss: 1.8422
Epoch [197/200], Loss: 1.9226
Epoch [198/200], Loss: 1.7423
Epoch [199/200], Loss: 1.8963
Epoch [200/200], Loss: 1.5288
```

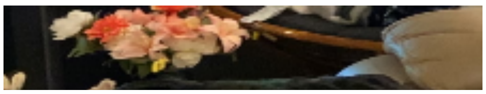
```
# Prediction 1
pred_img_path = '/content/drive/My Drive/Co
# image_size = 224  # Resize to this if nec

# Read and preprocess the image
pred_img_array = cv2.imread(pred_img_path)
pred_img_array = cv2.resize(pred_img_array,
pred_img_tensor = transforms.ToTensor()(pre

# Feed the model for prediction
with torch.no_grad():
    pred_val = model(pred_img_tensor)
    predicted_breed = sorted(new_list)[torc

from google.colab.patches import cv2_imshow
cv2_imshow(cv2.resize(cv2.imread(pred_img_p
# Display the predicted breed
print("Predicted Breed for this Dog is:", p

# Prediction 2
pred_img_path = '/content/drive/My Drive/Co
# image_size = 224  # Resize to this if nec

# Read and preprocess the image
pred_img_array = cv2.imread(pred_img_path)
pred_img_array = cv2.resize(pred_img_array,
pred_img_tensor = transforms.ToTensor()(pre

# Feed the model for prediction
with torch.no_grad():
    pred_val = model(pred_img_tensor)
    predicted_breed = sorted(new_list)[torc
```

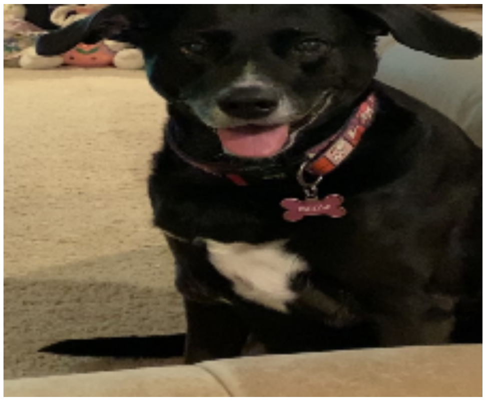
```
from google.colab.patches import cv2_imshow
cv2_imshow(cv2.resize(cv2.imread(pred_img_p
# Display the predicted breed
print("Predicted Breed for this Dog is:", p

# Prediction 3
pred_img_path = '/content/drive/My Drive/Co
# image_size = 224  # Resize to this if nec

# Read and preprocess the image
pred_img_array = cv2.imread(pred_img_path)
pred_img_array = cv2.resize(pred_img_array,
pred_img_tensor = transforms.ToTensor()(pre

# Feed the model for prediction
with torch.no_grad():
    pred_val = model(pred_img_tensor)
    predicted_breed = sorted(new_list)[torc

from google.colab.patches import cv2_imshow
cv2_imshow(cv2.resize(cv2.imread(pred_img_p
# Display the predicted breed
print("Predicted Breed for this Dog is:", p

# Prediction 4
pred_img_path = '/content/drive/My Drive/Co
# image_size = 224  # Resize to this if nec

# Read and preprocess the image
pred_img_array = cv2.imread(pred_img_path)
pred_img_array = cv2.resize(pred_img_array,
pred_img_tensor = transforms.ToTensor()(pre

# Feed the model for prediction
with torch.no_grad():
    pred_val = model(pred_img_tensor)
    predicted_breed = sorted(new_list)[torc

from google.colab.patches import cv2_imshow
cv2_imshow(cv2.resize(cv2.imread(pred_img_p
# Display the predicted breed
print("Predicted Breed for this Dog is:", p

# Prediction 5
```

```
pred_img_path = '/content/drive/My Drive/Co
# image_size = 224  # Resize to this if nec

# Read and preprocess the image
pred_img_array = cv2.imread(pred_img_path)
pred_img_array = cv2.resize(pred_img_array,
pred_img_tensor = transforms.ToTensor()(pre

# Feed the model for prediction
with torch.no_grad():
    pred_val = model(pred_img_tensor)
    predicted_breed = sorted(new_list)[torc

from google.colab.patches import cv2_imshow
cv2_imshow(cv2.resize(cv2.imread(pred_img_p
# Display the predicted breed
print("Predicted Breed for this Dog is:", p
```

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

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





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

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

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

Change runtime type