Dog Breed Classification Using Deep Learning

Mel Gerst
Department of Computing & Informatics, ITCS-5154
University of North Carolina, Charlotte
Charlotte, USA

Nov 24th, 2024

***Abstract*—This report explores deep learning and the use of neural networks to classify images. The task is to classify a dog's breed based on an image, using a trained neural network. In this paper, three different neural networks were explored including ResNet50v2, VGG and SimpleCNN. The models were all trained using a set of 20k dog images, with an 80/20 split for training and test data. Finally, each model was presented with 5 brand new images for breed prediction as a final test. As the paper describes, the results were quite varied based on the AI model chosen and ranged from strong performing (ResNet50v2) to very low performing (SimpleCNN).**

Key Words—Dog Breed Classification, Deep Learning, Neural Networks, Artificial Intelligence, ResNet50, VGG, SimpleCNN

I. Introduction

This report explores deep learning and the use of neural networks to classify images. The task is to classify a dog's breed based on an image, using a trained neural network.

Motivation

My motivation stems from prior work with barcode scanners and the current necessity to scan barcodes to identify products. I foresee a future time when barcodes are outdated, and most items can be identified using visual cues. As a result, I am excited to attempt this project of using a dog picture database and deep learning techniques and see if an algorithm can identify the breed of a random dog picture.

Additionally, I wanted to tackle a practical topic relatable to everyday life. I have several dogs and thought it would be fun to do a dog related project with pictures of my own dogs as prediction samples to test. The problem definition and my approach are outlined here.

Problem Statement

"Can a dog breed be visually identified simply from a picture using machine learning techniques, deep learning and a trained algorithm?"

Concise Summary of Solution

In short, three different neural networks were explored including ResNet50v2, VGG and SimpleCNN. The models were all trained using a set of 20k preprocessed and augmented dog images, with an 80/20 split chosen for training and test data. Finally, each trained model was presented with 5 brand new images for breed prediction as a final test. As this paper will describe, the results were quite varied based on the AI model chosen and ranged from strong performing (ResNet50v2) to very low performing (SimpleCNN).

Challenges

I expected this project to be challenging and there were certainly key decisions on the approach that were necessary.

The first challenge was to acquire enough image data to train and test the model. The data used was from "Dog's Breed Identification Dataset", a Kaggle[1] dataset containing over 20K dog images. These images were then split into training and testing sets for model validation.

The second challenge was to identify the best environment to support the computational models, dataset and resources to support the multiple test runs required. The model required large storage and computation power to run in a reasonable amount of time. Google Colab was chosen and a .ipynb Jupyter notebook was created to extract the dog images and build and test the model. Despite running on GPU with purchased GPU credits, the average run time was still ~1 hour per run. Both A100 and T4 GPUs were used during the execution.

Finally, it was time to decide which AI model to use to tackle the problem statement. The original project I duplicated used a ResNet50v2 pre-trained model. ResNet has been shown to perform well on classifying image data and the inspirational works I read showed solid performance. I also decided to contribute two additional models to test including a VGG pre-trained model and a SimpleCNN model. In this way, the models could be compared on training and prediction performance.

II. Background & Related Works

This paper was inspired by several works listed in the citations. After detailed review of all these papers, my confidence level was high that I could duplicate at least one of these works and achieve reasonable results.

My inspirational paper was "Dog Breed Identification using ResNet Model", Reddy, Y. A. ., Kumar, Y. S. ., M, S. ., & Mana, S. C. . (2023)[2]. This paper broke the problem down very well and cited strong results for classifying dog breeds based on images. In this paper the authors compared two CNN models including ResNet50v2 and Inception V3. They also built a flask web interface to capture dog photos for prediction. Based on their training results, they showed the ResNet50v2 model could achieve 91% training accuracy and the Inception V3 model achieved 80% accuracy with 100 epochs.

The project I chose to duplicate was "Dog's Breed Identification using Deep Learning", TechVidVan [3], Website - https://techvidvan.com/tutorials/dog-breed-classification/.

This website contained code snippets I could incorporate into my own version of the project and also used the ResNet50v2 AI model for classification. The author was able to achieve 80.5% training accuracy with only 20 epochs. The combination of these two sources gave me the confidence to tackle this project.

Additional inspiration came from the paper "Dog Breed Classification Using Deep Learning", Varshney, Akash & Katiyar, Abhay & Singh, Aman & Chauhan, Surendra (2021) [4] and "Novel Dataset for Fine-Grained Image Categorization", Aditya Khosla and Bangpeng Yao and Nityananda Jayadevaprakash and Li Fei-Fei [5]. These papers provided additional insight into the approach to implement, the quality of the dataset and insights into the results I was ultimately able to achieve.

Approaches Pros and Cons

A significant positive and major influence on my approach was the successful use of ResNet50v2 to execute the project with strong results. I also researched the general use of ResNet50 from other web sources regarding its use in image classification and the broad conclusions showed strong support for using ResNet50 for these types of problems.

A significant con, or challenge, to these approaches was the storage and processing of 20k dog images. This would require a significant amount of memory usage and processing power. As a result of this challenge, I chose to cut the dataset in half and only create a training dataset with roughly half the images covering 60 dog breeds, instead of the original 120 breeds. This roughly cut the processing and memory storage in half. To accelerate processing, as stated earlier, I executed the code on Google Colab with purchased GPU credits to further reduce processing time.

Relation to My Approach

My final approach for the ResNet50 model strongly leverages the code available from TechVidVan. I had to modify the code to download the images from my Google Drive account, where I uploaded the data for processing. I also added code to print the training accuracies of the models and to present the model with five images of my own, unseen by the trained model, to test the prediction effectiveness of the trained model's classification capability.

In addition, I contributed code to add two additional classification models including VGG and SimpleCNN. The contribution details and results of these implementations are further discussed in the following sections.

III. Method

My approach to implementing this project is broken down below and visually summarized in the architecture diagram.

Description of Method

1) Import the necessary libraries. The technology, languages and libraries used in this project included the following: Python, Tensorflow: 2.3.1, Opencv, Sklearn, Numpy, Pandas, Matplotlib, ResNet50v2, VGG, SimpleCNN.
2) Download images from my Google drive.
3) Cut original dataset of 120 dog breeds and 20k images in half to 60 dog breeds and ~10k images. This was done simply to save processing time and memory.
4) Encode, scale and normalize the images. The remaining images were preprocessed and augmented to aid in training including resizing all images to 224 x 224, convert to 3 color channels (RGB), converting to numeric data and scale to -1 to1.
5) The images were then split into a training dataset (80%) and test dataset (20%). The datasets were then augmented to create more robust training data via rotation, shifting, shearing, flipping and zooming.
6) Build the models (x 3)
7) Train the models (x 3)
8) Test the models (x 3)
9) Predict breed using five personal photos (x 3)
10) Collect the results (x 3)

Algorithms

I executed three different AI classifications during the execution of this project. These three models are described here.

ResNet50v2 – Residual Network (ResNet) is a CNN model which benefits from transfer learning, with pretraining using the Imagenet database. ResNet contains convolutional, pooling, activation and fully connected layers stacked next to each other. These layers of the ResNet are pre-trained on more than a million images from the ImageNet database. Resnet50V2 is 50 layers deep and applies batch normalization and a RELU activation function before the input is multiplied by convolutional operations(weight matrix). Due to the many layers, ResNet50v2 solves complex problems and increases model accuracy and performance.[3]

VGG – The Visual Geometry Group (VGG) AI model was developed by researchers at Oxford and is another pretrained CNN model. In this project I used VGG-16 which consists of 13 convolutional layers and three fully connected layers.

VGG's convolutional layers use an image sized 224 x 244 and leverage a minimal receptive field of 3×3. This is followed by a RELU layer. The convolution stride is fixed at 1 pixel.[6]

SimpleCNN – The SimpleCNN model is an untrained image classification model with convolutional layers, pooling layers and output layer. In this project I implemented a SimpleCNN model with 3 convolutional layers, 3 RELU layers, 3 pooling layers and an output layer. The convolutional layers used a 3x3 filter and stride of 1 pixel.

Architecture in a Diagram

To maintain consistency between models and experiments, I separated the execution into two major sections. The first section, shown below in green, was the importing of libraries and the dataset, preprocessing and augmenting the images, and creating the training and test datasets. With this approach, the resulting datasets could be used for all three AI models, saving processing time and ensuring the same training and test data was used for all three models. These steps are broken down below in steps 1-5 in Figure 1.

The second section of execution was focused on training, testing, predicting breeds and documenting the results. This was performed for all three models and broken down in steps 6-10 below in the diagram in Figure 1.

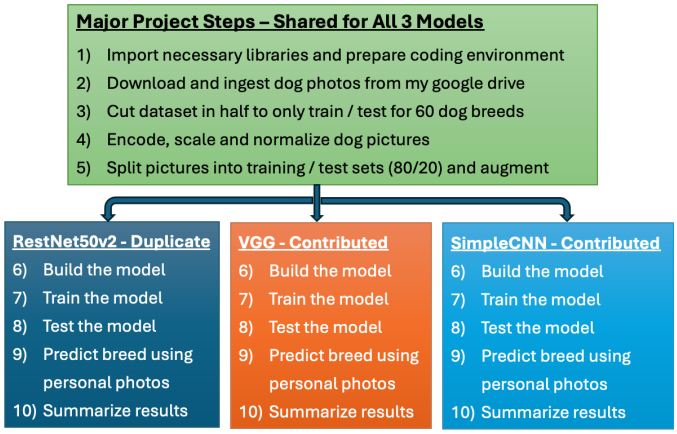

Figure 1

IV. Experiments

I experimented with all three algorithms including testing with varying amounts of epochs per solution. I also tested the predictions of all three algorithms with the same test pictures, shown below in Figure 2.

Figure 2

Test Results / Observations / Analysis

Shown below in Figure 3 is a summary of the results from the experiments.

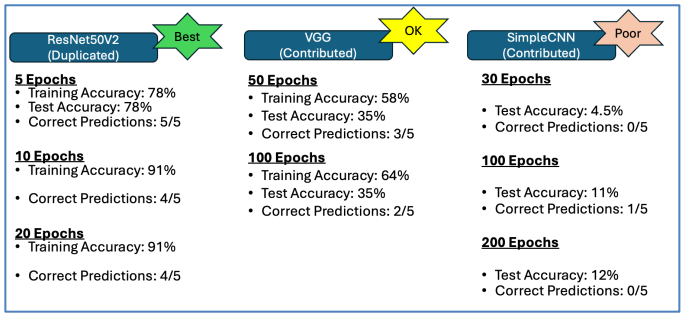

Figure 3

ResNet50v2 - Duplicated Model - The ResNet50v2 deep learning pre-trained model performed the best overall. Training accuracy improved from 78% to 91% from 5 epochs to 10, but did not improve over 91%. My best result using the ResNet50v2 model was a training accuracy of 91% using 10 epochs, and I achieved the same accuracy with 20 epochs.

Interestingly, the 5-epoch model with only 78% training accuracy correctly predicted all 5 dog breeds on the prediction test pictures. The 10/20 epoch models, with 91% accuracy, only correctly predicted 4 of 5 test pictures.

Comparing my results to the results from the surveyed papers, the TechVidVan[3] model which was duplicated achieved a training accuracy of 80.5% with 20 epochs. The motivation original paper "Dog Breed Identification using ResNet Model" Reddy, Y. A., Kumar, Y. S. ., M, S. ., & Mana, S. C. . (2023)[2] achieved an accuracy of 91% with 120 epochs

Contributed Models (VGG / SimpleCNN) – As the result summary in Figure 3 shows, my attempts to improve beyond the results of the ResNet50v2 model were unsuccessful. Even with increasing the number of epochs significantly, I was unable to match the results of the ResNet algorithm using the exact same training data.

The best results from the VGG pre-trained model were a training accuracy of 64%, test accuracy of 35% and 3 of 5 correct predictions. The VGG model does show promise and with enough experimentation I may be able improve to a higher accuracy. However, I believe it's not capable of matching the performance of ResNet50v2.

The best results of the SimpleCNN model were a test accuracy of 12% and 1 of 5 correct predictions. The SimpleCNN model appears to lack the learning horsepower for analyzing the complex visuals in three channels (RGB) or the 20k images are not enough for effective training.

V. Conclusion

I learned a great deal from executing this project and comparing the three different algorithms. I was also able to reach several conclusions.

Summary of Conclusions

- The choice of AI model is critical to the results. The same preprocessed pictures were used in all three models, but the models had widely different performance results.
- Using a pre-trained model is valuable. The training accuracy was much higher using the ImageNet pre-trained models Resnet50v2 & VGG.
- In general, more training epochs did produce higher training accuracies. The ResNet model greatly benefitted from additional training, but the VGG and SimpleCNN models only improved slightly.
- Importantly, the benefits of training with additional epochs eventually tapered off with very little to no benefit of further increasing the number of epochs.
- The prediction capability of the models generally increased with higher training accuracies but still allowed for some variability. For example, the Resnet50v2 model with a training accuracy of 91% was not able to correctly predict all 5 pictures reliably, but when trained with only 5 epochs and a training accuracy of 78% was able to correctly predict all five pictures.

Concluding Remarks

I enjoyed executing this project and found it to be very valuable. The experimentation with three different models to perform the same task was insightful and I personally witnessed the value of leveraging pretrained models and impact of training with different epoch levels.

With enough training I showed that a model could become a powerful tool for analyzing and categorizing images. However, I also showed that reaching 100% prediction capability and eliminating all prediction errors would be extremely difficult or perhaps impossible. This observation is an important aspect to working with artificial intelligence. Limitations of AI accuracy is a trending topic in the media and a growing concern for companies as they attempt to further integrate AI into their everyday business operations. It will be critical in the future for the users of AI to understand its limitations and put in place checks and balances for business processes to accommodate these limitations.

VI. My Contributions

The code base and the results for the test runs for this project are contained in the following public Githib directory and shown in the appendix:

https://github.com/mgerst1/ITCS-5154-MelGerst

IMPORTANT NOTE TO GRADER / TA – The code base relies on importing the training data from my personal Google Drive. This was necessary since Github would not allow the 20k images to be uploaded to this Github account, so the code cannot be executed as is without accessing my Google drive.

Explanation of Contributions

The code can be split into three major sections aligned to the AI models deployed, namely Resnet50, VGG, and SimpleCNN. My contributions are detailed below for these sections.

ResNet50: This code is largely duplicated from the TechVidVan[3] webpage, available at the link below.

Website-https://techvidvan.com/tutorials/dog-breed-classification/

My contributions included transferring the code into a Google Colab Jupyter notebook and piecing it together. I also uploaded the Kaggle dataset to my Google Drive. I then contributed code to access the dataset from my google drive and storing it into variables. I also contributed code to store the final training array back onto my Google drive to save time during multiple testing runs by commenting out the image processing section of code and loading the resulting array into memory for each test run. I added additional code to capture the training and test accuracies. I also contributed code to load 5 different prediction test pictures and then print out the final resulting predictions.

VGG / SimpleCNN: For these two models / sections I contributed the code, with support from generative AI, to import and define these models, set the model parameters, and tie the models to the previously used variables used earlier in the code base. The contributed code then trains and tests the models and captures the accuracy results. Both models are then used to predict the breeds for the same five test photos and the predicted breed results are output.

Further, significant time was spent training and experimenting with the three models, including varying the epoch ranges, to capture the results and record them. As stated earlier, each test run typically required 1 hour of execution time running on Google Colab GPUs with purchased GPU credits.

VII. References & Citations

[1] Dog's Breed Identification Dataset, https://www.kaggle.com/c/dog-breed-identification
Kaggle Citation: @misc{dog-breed-identification, author = {Will Cukierski}, title = {Dog Breed Identification}, publisher = {Kaggle}, year = {2017}, url = {https://kaggle.com/competitions/dog-breed-identification} }

[2] "Dog Breed Identification using ResNet Model", Reddy, Y. A. ., Kumar, Y. S. ., M, S. ., & Mana, S. C. . (2023)., International Journal on Recent and Innovation Trends in Computing and Communication, 11(7s),64–71, https://doi.org/10.17762/ijritcc.v11i7s.6977

[3] "Dog's Breed Identification using Deep Learning", TechVidVan, Website - https://techvidvan.com/tutorials/dog-breed-classification/

[4] "Dog Breed Classification Using Deep Learning", Varshney, Akash & Katiyar, Abhay & Singh, Aman & Chauhan, Surendra. (2021), Dog Breed Classification Using Deep Learning. 1-5. 10.1109/CONIT51480.2021.9498338.

[5] "Novel Dataset for Fine-Grained Image Categorization", Aditya Khosla and Bangpeng Yao and Nityananda Jayadevaprakash and Li Fei-Fei, First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition,2011,June, Colorado Springs, CO, @inproceedings{khosla_fgvc2011}

[6] "Very Deep Convolutional Networks (VGG) Essential Guide", Gaudenz Boesch, October 6, 2021, Website - https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks/

A. Appendix – Code

```
# -*- coding: utf-8 -*-
"""DogBreedClassifier-MelGerst2.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1Oo1ThysfOEmHutnf-6g_kia8--_kdP-n

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

# 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/DogBreedClassifierProject/AI-model/labels.csv")
#store training and testing images folder location
training_data = '/content/drive/My Drive/ColabNotebooks/DogBreedClassifierProject/AI-data/train/'
testing_data =  '/content/drive/My Drive/ColabNotebooks/DogBreedClassifierProject/AI-data/test/'

# 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))

# 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.breed.unique()))
print("The breeds used for training and testing are:", sorted(df_labels.breed.unique()))

# Create a numpy array of the shape (number of dataset records, image size , image size, 3 for rgb channel layer)
# Input for model
train_x = np.zeros((len(df_labels), image_size, image_size, 3), dtype='float32')

#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,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.float32)).copy(), axis=0))
  # Update the train_x variable with new element
  train_x[i] = img_array

print(train_x.shape)
np.save('/content/drive/My Drive/ColabNotebooks/DogBreedClassifierProject/AI-data/file1.npy', train_x)

train_x = np.load('/content/drive/My Drive/ColabNotebooks/DogBreedClassifierProject/AI-data/file1.npy')
print(train_x.shape)

# 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,random_state=42)

#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)

# Model #1 - 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', include_top=False)
# 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)

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

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

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

# Get the image of the dog #1 for prediction
```

```
pred_img_path = '/content/drive/My Drive/ColabNotebooks/DogBreedClassifierProject/AI-model/rottweiler.jpg'
# 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,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].astype(np.float32)).copy(), axis=0))

# 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_size))))
# 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/AI-model/toby.jpg'
# 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_size,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_array2 = preprocess_input(np.expand_dims(np.array(pred_img_array2[...,::-1].astype(np.float32)).copy(), axis=0))

# 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,image_size))))
# 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/AI-model/bella.jpg'
# 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_size,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_array3 = preprocess_input(np.expand_dims(np.array(pred_img_array3[...,::-1].astype(np.float32)).copy(), axis=0))

# 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,image_size))))
# 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/AI-model/irishwolfhound.jpg'
# 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_size,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_array4 = preprocess_input(np.expand_dims(np.array(pred_img_array4[...,::-1].astype(np.float32)).copy(), axis=0))

# 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,image_size))))
# 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/AI-model/whippet.jpg'
# 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_size,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_array5 = preprocess_input(np.expand_dims(np.array(pred_img_array5[...,::-1].astype(np.float32)).copy(), axis=0))

# 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,image_size))))
#Display the predicted breed of dog
predicted_breed5 = sorted(new_list)[np.argmax(pred_val5)]
print("Predicted Breed for this Dog is :",predicted_breed5)
print("Check image size: ", image_size)
```

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

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

# Assuming train_y is one-hot encoded, if not, you might need to do this
train_y = tf.keras.utils.to_categorical(train_y, num_classes=60)

# Split the data
x_train, x_test, y_train, y_test = train_test_split(train_x, train_y, test_size=0.2, random_state=42)

def create_vgg_model(input_shape, num_classes):
    base_model = tf.keras.applications.VGG16(weights='imagenet', include_top=False, input_shape=input_shape)

    # 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, activation='softmax')
    ])

    return model, base_model # Return both model and base_model

input_shape = (224, 224, 3)
num_classes = 60
model, base_model = create_vgg_model(input_shape, num_classes) # Get both returned values

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

history = model.fit(x_train, y_train,
                    epochs=50,
                    batch_size=32,
                    validation_data=(x_test, y_test))

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

base_model.trainable = True  # Now base_model is accessible here
for layer in base_model.layers[:-4]:  # Unfreeze the last few layers
    layer.trainable = False

model.compile(optimizer=tf.keras.optimizers.Adam(1e-5),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

history_finetune = model.fit(x_train, y_train,
                             epochs=10,
                             batch_size=32,
                             validation_data=(x_test, y_test))

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

# Second round of predictions with VGG model
print("Check image size: ", image_size)
# Get the image of the dog #1 for prediction
pred_img_path = '/content/drive/My Drive/ColabNotebooks/DogBreedClassifierProject/AI-model/rottweiler.jpg'
# 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,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].astype(np.float32)).copy(), axis=0))

# 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_size))))
# 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/AI-model/toby.jpg'
# 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_size,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_array2 = preprocess_input(np.expand_dims(np.array(pred_img_array2[...,::-1].astype(np.float32)).copy(), axis=0))
```

```
# 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,image_size))))
# 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/AI-model/bella.jpg'
# 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_size,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_array3 = preprocess_input(np.expand_dims(np.array(pred_img_array3[...,::-1].astype(np.float32)).copy(), axis=0))

# 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,image_size))))
# 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/AI-model/irishwolfhound.jpg'
# 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_size,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_array4 = preprocess_input(np.expand_dims(np.array(pred_img_array4[...,::-1].astype(np.float32)).copy(), axis=0))

# 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,image_size))))
# 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/AI-model/whippet.jpg'
# 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_size,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_array5 = preprocess_input(np.expand_dims(np.array(pred_img_array5[...,::-1].astype(np.float32)).copy(), axis=0))

# 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,image_size))))
#Display the predicted breed of dog
predicted_breed5 = sorted(new_list)[np.argmax(pred_val5)]
print("Predicted Breed for this Dog is :",predicted_breed5)

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

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

# Assume train_x and train_y are defined
# train_x: numpy array of shape (5175, 224, 224, 3)
# train_y: numpy array of shape (5175,)

# Split the dataset
# x_train, x_test, y_train, y_test = train_test_split(train_x, train_y, test_size=0.2, random_state=42)

# Set batch size
# batch_size = 64

# Define data augmentation and normalization transformations
train_transforms = transforms.Compose([
    transforms.ToPILImage(),  # Convert numpy array to PIL Image
    transforms.RandomRotation(45),
    transforms.RandomAffine(degrees=0, translate=(0.2, 0.2)),
    transforms.RandomResizedCrop(size=(224, 224), scale=(0.75, 1.0)),
    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, transform=None):
        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) # get the class indices from one-hot encoded y_train
y_test_indices = np.argmax(y_test, axis=1) # get the class indices from one-hot encoded y_test
train_dataset = CustomDataset(x_train, y_train_indices, transform=train_transforms) # use class indices for y_train
test_dataset = CustomDataset(x_test, y_test_indices, transform=test_transforms) # use class indices for y_test

# Create DataLoaders
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)




# Define the SimpleCNN model
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)  # Input: 3 channels (RGB)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)  # Pooling layer
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)

        # Calculate the input size for the first fully connected layer
        self.fc1_input_size = 128 * (224 // 8) * (224 // 8)  # After three pooling layers
        self.fc1 = nn.Linear(self.fc1_input_size, 256)
        self.fc2 = nn.Linear(256, 60)  # Output layer for 60 classes

    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 the output
        x = self.fc1(x)
        x = nn.ReLU()(x)
        x = self.fc2(x)
        return nn.LogSoftmax(dim=1)(x)

# Initialize model, loss function, and optimizer
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = SimpleCNN().to(device)
criterion = nn.NLLLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

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

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

    print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {loss.item():.4f}')

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

with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        test_loss += criterion(outputs, labels).item()  # Add to total loss
        pred = outputs.argmax(dim=1)  # Get the predicted class
        correct += (pred == labels).sum().item()  # Count correct predictions

test_accuracy = correct / len(test_dataset)
print(f'Test Loss: {test_loss/len(test_loader):.4f}, SimpleCNN Test Accuracy: {test_accuracy:.4f}')

# Prediction 1
pred_img_path = '/content/drive/My Drive/ColabNotebooks/DogBreedClassifierProject/AI-model/rottweiler.jpg'
# image_size = 224  # Resize to this if necessary

# Read and preprocess the image
pred_img_array = cv2.imread(pred_img_path)
pred_img_array = cv2.resize(pred_img_array, (image_size, image_size))
pred_img_tensor = transforms.ToTensor()(pred_img_array).unsqueeze(0).to(device)  # Add batch dimension

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

from google.colab.patches import cv2_imshow
cv2_imshow(cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((image_size,image_size))))
# Display the predicted breed
print("Predicted Breed for this Dog is:", predicted_breed)

# Prediction 2
pred_img_path = '/content/drive/My Drive/ColabNotebooks/DogBreedClassifierProject/AI-model/toby.jpg'
# image_size = 224  # Resize to this if necessary

# Read and preprocess the image
pred_img_array = cv2.imread(pred_img_path)
pred_img_array = cv2.resize(pred_img_array, (image_size, image_size))
pred_img_tensor = transforms.ToTensor()(pred_img_array).unsqueeze(0).to(device)  # Add batch dimension

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

from google.colab.patches import cv2_imshow
cv2_imshow(cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((image_size,image_size))))
# Display the predicted breed
print("Predicted Breed for this Dog is:", predicted_breed)

# Prediction 3
pred_img_path = '/content/drive/My Drive/ColabNotebooks/DogBreedClassifierProject/AI-model/bella.jpg'
# image_size = 224  # Resize to this if necessary

# Read and preprocess the image
pred_img_array = cv2.imread(pred_img_path)
pred_img_array = cv2.resize(pred_img_array, (image_size, image_size))
pred_img_tensor = transforms.ToTensor()(pred_img_array).unsqueeze(0).to(device)  # Add batch dimension

# Feed the model for prediction
```

```
with torch.no_grad():
    pred_val = model(pred_img_tensor)
    predicted_breed = sorted(new_list)[torch.argmax(pred_val).item()]

from google.colab.patches import cv2_imshow
cv2_imshow(cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((image_size,image_size))))
# Display the predicted breed
print("Predicted Breed for this Dog is:", predicted_breed)

# Prediction 4
pred_img_path = '/content/drive/My Drive/ColabNotebooks/DogBreedClassifierProject/AI-model/irishwolfhound.jpg'
# image_size = 224  # Resize to this if necessary

# Read and preprocess the image
pred_img_array = cv2.imread(pred_img_path)
pred_img_array = cv2.resize(pred_img_array, (image_size, image_size))
pred_img_tensor = transforms.ToTensor()(pred_img_array).unsqueeze(0).to(device)  # Add batch dimension

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

from google.colab.patches import cv2_imshow
cv2_imshow(cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((image_size,image_size))))
# Display the predicted breed
print("Predicted Breed for this Dog is:", predicted_breed)

# Prediction 5
pred_img_path = '/content/drive/My Drive/ColabNotebooks/DogBreedClassifierProject/AI-model/whippet.jpg'
# image_size = 224  # Resize to this if necessary

# Read and preprocess the image
pred_img_array = cv2.imread(pred_img_path)
pred_img_array = cv2.resize(pred_img_array, (image_size, image_size))
pred_img_tensor = transforms.ToTensor()(pred_img_array).unsqueeze(0).to(device)  # Add batch dimension

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

from google.colab.patches import cv2_imshow
cv2_imshow(cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((image_size,image_size))))
# Display the predicted breed
print("Predicted Breed for this Dog is:", predicted_breed)
```