Dog Breed Classifier using CNN Project

1. Project Overview

Computer Vision is an interdisciplinary field that deals with how computers can be made to gain high-level understanding from digital images or videos. From a technical perspective, it seeks to automate tasks that the human visual system can perform.

In this project, I will use Computer Vision to solve the problem of identifying dog breeds. Specifically, if a dog is detected in the image, it will provide an estimate of the dog's breed. If a human is detected, it will provide an estimate of the dog breed that is most resembling.

2. Problem Statement

The goal was to create a web-based breed classification program; The related duties are as follows:

i. Download and pre-process human face and dog breed data.
ii. Train a classifier that can determine if an image contains a human face.
iii. Train a classifier that can determine if an image resembles a dog.
iv. Train a classifier that identifies dog breeds based on images.
v. Build a webapp to deploy the program on the local server.

The final application is expected to be useful for dog breed classification, which is convenient for users and has high accuracy.

3. Metrics

Accuracy is a common metric for binary classifiers; it takes into account both true positives and true negatives with equal weight.

accuracy = (true positives + true negatives)/ dataset size

This metric was used when evaluating the classifier because false negatives and false positives both erode the user experience:

- False negatives result in either a longer delay between the user pointing the camera text and device speaking the text ("processing delay") or in the worst case, completely prevent the application from reading said text.
- On the other hand, false positives make the application try to extract text from images that don't contain any. This results in unnecessary computations on the remote server, which can be both costly and slow the application down. In the worst case, this might also result in the application reading gibberish.

4. Data Understanding

Next, we will learn about existing data sets.

The dataset provided includes a human dataset with 13233 images and a dog dataset with 8351 images. Each image in the dog dataset is labeled with one of 133 dog breeds. For efficient breed classification, the images in the breed dataset are divided into training sets, evaluators and test sets with an approximate ratio of 8: 1: 1.

The image below shows the amount of data obtained for each dog breed in the training dataset.

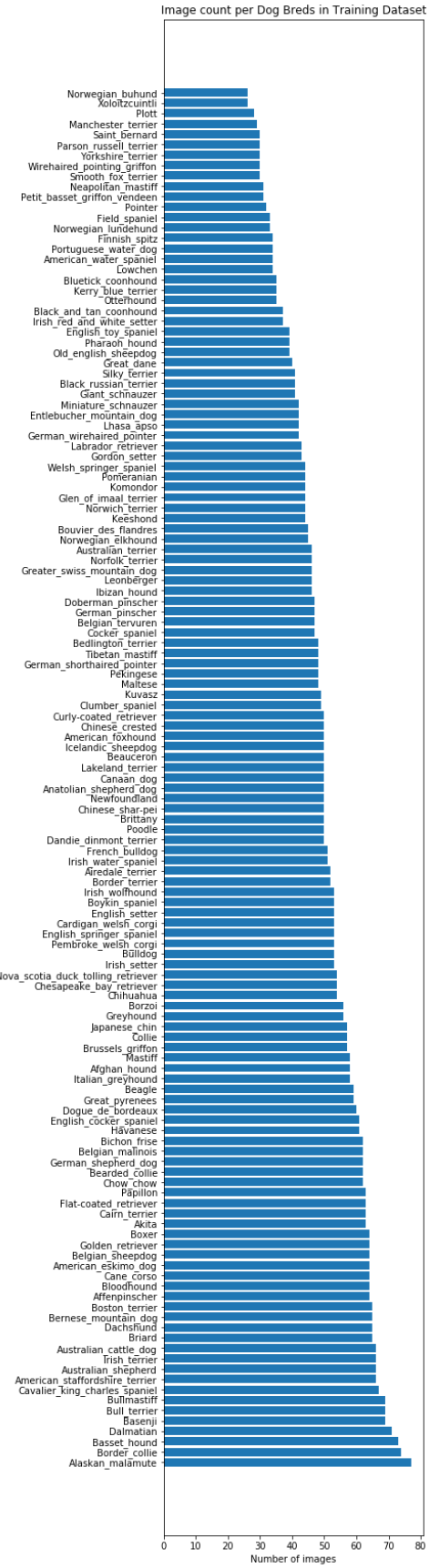
Image count per Dog Breds in Training Dataset
Norwegian_buhund
Xoloitzcuintli
Plott
Manchester_terrier
Saint_bernard
Parson_russell_terrier
Yorkshire_terrier
Wirehaired_pointing_griffon
Smooth_fox_terrier
Neapolitan_mastiff
Petit_basset_griffon_vendeen
Pointer
Field_spaniel
Norwegian_lundehund
Finnish_spitz
Portuguese_water_dog
American_water_spaniel
Cowchen
Bluetick_coonhound
Kerry_blue_terrier
Otterhound
Black_and_tan_coonhound
Irish_red_and_white_setter
English_toy_spaniel
Pharaoh_hound
Old_english_sheepdog
Great_dane
Silky_terrier
Black_russian_terrier
Giant_schnauzer
Miniature_schnauzer
Entlebucher_mountain_dog
Lhasa_apso
German_wirehaired_pointer
Labrador_retriever
Gordon_setter
Welsh_springer_spaniel
Pomeranian
Komondor
Glen_of_imaal_terrier
Norwich_terrier
Keeshond
Bouvier_des_flandres
Norwegian_elkhound
Australian_terrier
Norfolk_terrier
Greater_swiss_mountain_dog
Leonberger
Ibizan_hound
Doberman_pinscher
German_pinscher
Belgian_tervuren
Cocker_spaniel
Bedlington_terrier
Tibetan_mastiff
German_shorthaired_pointer
Pekingese
Maltese
Kuvasz
Clumber_spaniel
Curly-coated_retriever
Chinese_crested
American_foxhound
Icelandic_sheepdog
Beauceron
Lakeland_terrier
Canaan_dog
Anatolian_shepherd_dog
Newfoundland
Chinese_shar-pei
Brittany
Poodle
Dandie_dinmont_terrier
French_bulldog
Irish_water_spaniel
Airedale_terrier
Border_terrier
Irish_wolfhound
Boykin_spaniel
English_setter
Cardigan_welsh_corgi
English_springer_spaniel
Pembroke_welsh_corgi
Bulldog
Irish_setter
Nova_scotia_duck_tolling_retriever
Chesapeake_bay_retriever
Chihuahua
Borzoi
Greyhound
Japanese_chin
Collie
Brussels_griffon
Mastiff
Afghan_hound
Italian_greyhound
Beagle
Great_pyrenees
Dogue_de_bordeaux
English_cocker_spaniel
Havanese
Bichon_frise
Belgian_malinois
German_shepherd_dog
Bearded_collie
Chow_chow
Papillon
Flat-coated_retriever
Cairn_terrier
Akita
Boxer
Golden_retriever
Belgian_sheepdog
American_eskimo_dog
Cane_corso
Bloodhound
Affenpinscher
Boston_terrier
Bernese_mountain_dog
Dachshund
Briard
Australian_cattle_dog
Irish_terrier
Australian_shepherd
American_staffordshire_terrier
Cavalier_king_charles_spaniel
Bullmastiff
Bull_terrier
Basenji
Dalmatian
Basset_hound
Border_collie
Alaskan_malamute
0
10
20
30
40
50
60
70
80
Number of images

It can be seen to some extent that, the data set is still somewhat asymmetric but not significant enough to cause classification problems. Better yet, the background of the object dataset is quite diverse, which means that the model will be provided with an unbiased data set and can be tested accurately on many different types of images. For example, the images below are taken from the dataset:

American_staffordshire_terrier

Dalmatian

5. Data Preprocessing

The images needed to be in 4D to use Keras; have shape (number of samples, row size of image in pixels, image column size in pixels, number of color channels). I created a function that loads an image and resizes it to a square image with a width of 224 pixels and a height of 224 pixels. Next, the image is converted to an array, which is then resized to a 4D tensor. The path_to_tensor function then stacks the returned images into a 4D tensor with the number of images from the training, validation, or test dataset depending on which image path is called.

```
from keras.preprocessing import image
from tqdm import tqdm

def path_to_tensor(img_path):
    # loads RGB image as PIL.Image.Image type
    img = image.load_img(img_path, target_size=(224, 224))
    # convert PIL.Image.Image type to 3D tensor with shape (224, 224, 3)
    x = image.img_to_array(img)
    # convert 3D tensor to 4D tensor with shape (1, 224, 224, 3) and return 4D tensor
    return np.expand_dims(x, axis=0)

def paths_to_tensor(img_paths):
    list_of_tensors = [path_to_tensor(img_path) for img_path in tqdm(img_paths)]
    return np.vstack(list_of_tensors)
```

6. Implementation

a. Create a human face detector

i. To detect human faces in images, OpenCV implementing Haar feature-based cascade classifier was used. This is a machine learning-based approach, where the stratification function is trained from a lot of positive and negative images, and then used to detect objects in other images. OpenCV provides many pre-trained face detection engines. Below is my code that initializes Haar Class Classifier from OpenCV and then generates face detection function to predict if input image contains human face or not:

```python
# returns "True" if face is detected in image stored at img_path
def face_detector(img_path):
    img = cv2.imread(img_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray)
    return len(faces) > 0
```

ii. ii. Although the function does not achieve 100% accuracy, it still gives acceptable performance. More specifically, this function achieves 100% of human face recognition images and 11% of dog face recognition images.

b. Create a dog detector

i. I use a pre-trained ResNet-50 model to detect dogs in images. The weights of this model have been trained on ImageNet, a very large, very popular dataset used for image classification and other vision tasks. ImageNet contains over 10 million URLs, each linking to an image containing an object from one of 1000 categories.

ii. While looking at the dictionary of ResNet-50 predict labels, you will notice that the categories corresponding to dogs appear in an uninterrupted sequence and correspond to dictionary keys 151-268, inclusive, to include all categories from 'Chihuahua' to 'Mexican hairless'. Thus, in order to check to see if an image is predicted to contain a dog by the pre-trained ResNet-50 model, we need only check if the ResNet50 predict labels returns a value between 151 and 268 (inclusive).

iii. iii. The accuracy of this function is very high, having detected 0% of human images with a detected dog and 100% of dog images with a detected dog.

c. Create a dogs breed classification model:

i. For testing purposes, I built a basic CNN network with Keras and Tensorflow, the network structure is as shown below, and the result achieved is only 13.5167% when performing the prediction. on the test set. This is expected because the network structure used is too simple and the number of training sessions is too small to fine-tune the weight of the model. Below is the code and summary model I made:

```
model = Sequential()

### TODO: Define your architecture.
model.add(Conv2D(filters = 16,
                 kernel_size = (3,3),
                 input_shape = (224,224,3),
                 activation='relu',strides=1))

model.add(Conv2D(filters = 32,
                 kernel_size = (3,3),
                 activation='relu',strides=1))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(filters = 32,
                 kernel_size = (3,3),
                 activation='relu',strides=1))
model.add(Conv2D(filters = 32,
                 kernel_size = (3,3),
                 activation='relu',strides=1))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(filters = 64,
                 kernel_size = (3,3),
                 activation='relu',strides=1))
model.add(Conv2D(filters = 64,
                 kernel_size = (3,3),
                 activation='relu',strides=1))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(GlobalAveragePooling2D())
model.add(Dense(133))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_46 (Conv2D)           (None, 222, 222, 16)      448
_________________________________________________________________
conv2d_47 (Conv2D)           (None, 220, 220, 32)      4640
_________________________________________________________________
max_pooling2d_31 (MaxPooling (None, 110, 110, 32)      0
_________________________________________________________________
conv2d_48 (Conv2D)           (None, 108, 108, 32)      9248
_________________________________________________________________
conv2d_49 (Conv2D)           (None, 106, 106, 32)      9248
_________________________________________________________________
max_pooling2d_32 (MaxPooling (None, 53, 53, 32)        0
_________________________________________________________________
conv2d_50 (Conv2D)           (None, 51, 51, 64)        18496
_________________________________________________________________
conv2d_51 (Conv2D)           (None, 49, 49, 64)        36928
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 24, 24, 64)        0
_________________________________________________________________
global_average_pooling2d_8 ( (None, 64)                0
_________________________________________________________________
dense_7 (Dense)              (None, 133)               8645
=================================================================
Total params: 87,653
Trainable params: 87,653
Non-trainable params: 0
_________________________________________________________________
```

ii. To increase the accuracy according to the actual needs of the problem without spending too much time on training, I use the pre-trained VGG-16 sample, provided by Udacity as a starting point. By training this model on board and the corresponding validator, the model was able to correctly classify dog breeds on 44,0191% of the test set. It's a big improvement over the CNN network I originally built myself, but it could be further refined for better accuracy. The code for my model build and its summary is as follows:

```
VGG16_model = Sequential()
VGG16_model.add(GlobalAveragePooling2D(input_shape=train_VGG16.shape[1:]))
VGG16_model.add(Dense(133, activation='softmax'))

VGG16_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_9 ( (None, 512)               0
_________________________________________________________________
dense_8 (Dense)              (None, 133)               68229
=================================================================
Total params: 68,229
Trainable params: 68,229
Non-trainable params: 0
_________________________________________________________________
```

iii. Next, I decided to use the ResNet-50 model as a new starting point. By going through the same process as with the VGG-16 model, the ResNet-50 model was able to correctly classify dog breeds on 81.221% of the tests. This is a great point to hit, so I didn't refine the model further. The code for my model

build and its summary is as follows:

```
### TODO: Define your architecture.
Resnet50_model = Sequential()
Resnet50_model.add(GlobalAveragePooling2D(input_shape=train_Resnet50.shape[1:]))
Resnet50_model.add(Dense(133, activation='softmax'))
Resnet50_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_3 ( (None, 2048)              0
_________________________________________________________________
dense_3 (Dense)              (None, 133)               272517
=================================================================
Total params: 272,517
Trainable params: 272,517
Non-trainable params: 0
_________________________________________________________________
```

d. Combine features and build a breed recognition program

i. The final step is to build an algorithm that takes an image path as input and determines whether a dog or a person is detected in the image.
ii. If a dog is detected in the image, the algorithm predicts that dog breed using the ResNet-50 transfer learning model.
iii. If a human is detected in the image, the algorithm calls that name out and predicts a dog breed using the ResNet-50 transfer learning model.
iv. If neither is detected in the image, the algorithm will call that out.
v. The algorithm as introduced below is built to output the provided image and its corresponding prediction.

```
### TODO: Write your algorithm.
### Feel free to use as many code cells as needed.
def main(file_path):
    is_human = face_detector(file_path)
    if dog_detector(file_path) or face_detector(file_path):
        return predict_breed(file_path)
    else:
        return "This image not have any dog or human."
```

7. Model Evaluation and Validation

a. The following is the exact degree to which each person received the model on the test set:
   i. Basis CNN model: 13.5167%
   ii. Transfer learning from VGG-16: 44.0191%
   iii. Transfer learning from ResNet-50: 81.221%
b. Initially, I denied my thinking that these accuracy differences were due to the different number of parameters in each model after seeing the number of parameters used to train the prediction model. guess. Specifically from the summary of the existing models:
   i. Basis CNN model: 87653 params
   ii. Transfer learning from VGG-16: 68229 params
   iii. Transfer learning from ResNet-50: 272517 params
c. However, after further research I realized that, the number of calculated params is the params that are tuned for the new dataset, while the actual number of params of the VGG-16 and ResNet-50 architectures is large. more. Moreover, both the weight sets of VGG-16 and ResNet-50 used above have been trained with a dataset of much larger

size than the current dataset, so the accuracy and calculation The generalization of these two sets of weights is superior to that of the CNN architecture that I built myself.

8. **Reflection**
    a. The goal of this project is to create a CNN with more than 60% test accuracy while training on GPU.
    b. In my opinion, this project is a great starting point to understand and practice deep learning. Even so, exploratory data analysis and data visualization are still important keys in creating and training effective machine learning models as these steps help in selecting an appropriate performance metric to evaluate model.
    c. Building CNN models from scratch is made extremely simple in Keras. However, this process is computationally expensive and time-consuming, especially without a GPU. This is one of the reasons why one of the many pre-trained models available in Keras and trained on the ImageNet dataset was used for forward learning. This significantly increases the experimental accuracy of the model when using the VGG-16 network first and then the ResNet-50 network.
9. **Improvement**

    Some ways to improve the algorithm in my opinion are as follows:
    a. Use a more suitable model for face recognition; such as OpenCV's DNN module or VGG Face.
    b. Increase the training data and possibly include a classifier of mixed breeds to improve model accuracy. This can lead to multiple classifiers and make the model more flexible.