Capstone Project

Rui Duan

Machine Learning Engineer
Nanodegree

Feb 18, 2021

Dog Breed Classifier

Definition

Project Overview

The Federacion Cynologique Internationale (FCI) has published over 300 dog breeds.
http://www.fci.be/en/Nomenclature/

It is hard to tell the dog breed given the look of a dog even for the dog lovers. Many people will be interested to find out the breed of the dog that they have seen from social networks.

The project is to build a dog breed classification app to identify a breed of dog by a dog image. If a human image is supplied, it will identify the resembling dog breed.

Two datasets have been provided by Udacity. They will be used for training/testing/validation.

- dog dataset
- human dataset

Problem Statement

This is a classification problem to identify dog breeds from dog images. In addition to it, it will find the closest resembling dog breed for human images.

The rule is simple: supply an image of human faces or dogs (face or full body). The model will detect if the image is human or dog. Then predict the dog breed from the supplied image.

There are three problems that the project will need to address.

1. Detect if the image contains a dog.
2. Detect if the image contains a human face.
3. Identify the dog breed by either the dog image or human face image

Two CNN models are built to classify dog breeds:

1. Build from scratch model
2. Build from pre-trained model using Transfer Learning

Metrics

The models are evaluated simply by the test accuracy from the datasets.

The dataset has been splitted to training, validation and testing sets. The testing set is used to measure the model performance. It is calculated as:

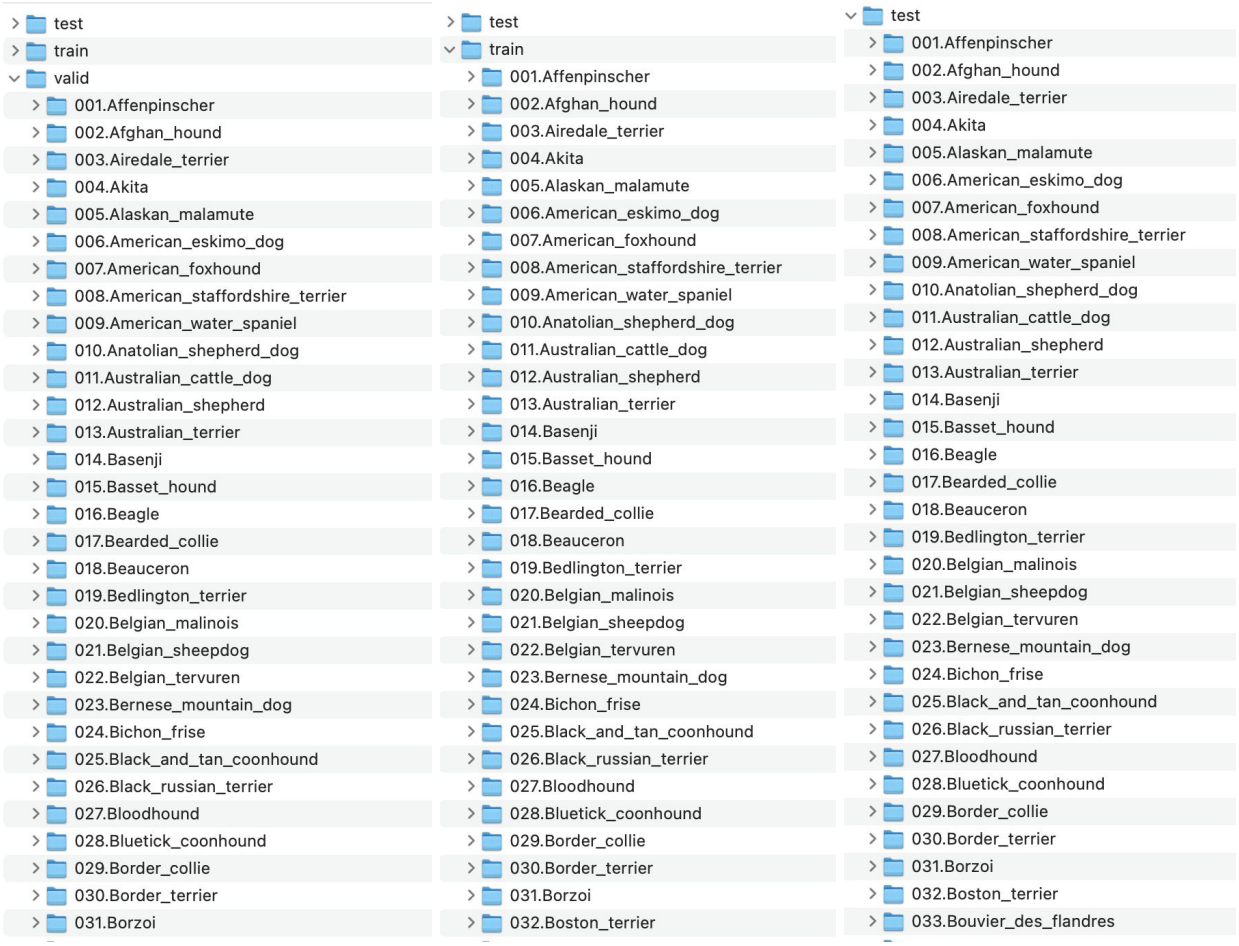
Accuracy=Number of items correctly classified/ All classified items

Analysis

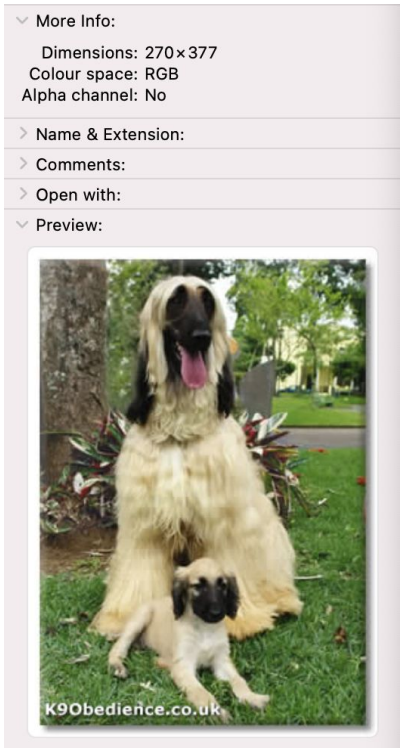
Data Exploration

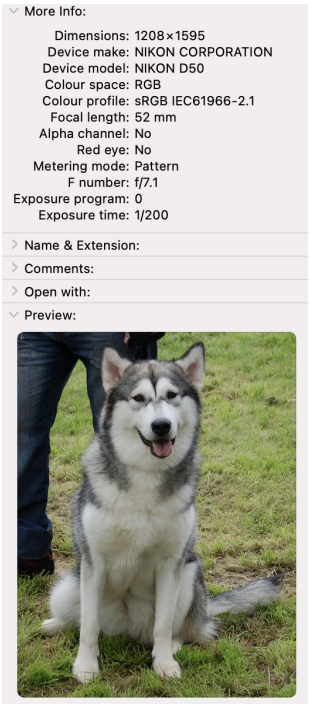
Dog dataset

In the dog dataset, there are 3 directories: train(6814 images), test(970 images) and valid(969 images). All of them contain 133 dog breeds folder of dog images.



They are in different sizes and different backgrounds. Most of them are around 200-600 pixels but there are some much larger images (1000+ pixels).





The images are from different angles of the dogs. All of them have dog faces. Below are some examples:

Silky Terrier



Papillon

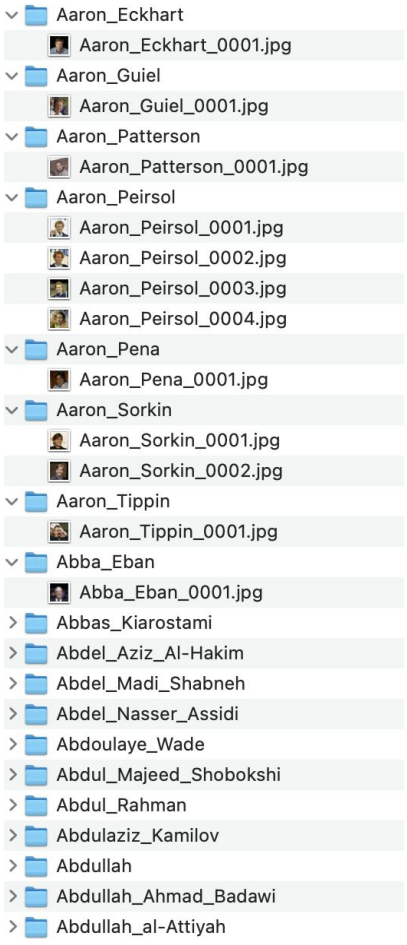


Kuvasz

Miniature Schnauzer


Human dataset

In the human dataset, there are 18983 images from 5749 people with the images in the folders with their names. Some of them have one image while others have multiple images.



All of human images are the same dimensions (250x250). Some of them have partial faces from other people.

Algorithms and Techniques

Convolutional Neural Networks(CNN) will be used to build the dog breed identification app. The model will be used as the backend of the web or mobile app.

The app will accept user supplied images and detect if it is a dog or human and provide the estimated dog breed.

To detect the images, a computer vision algorithm will be used.

Human face detection

In this project, the goal is to detect whether the supplied image contains a human face.

OpenCV's implementation of Haar feature-based cascade classifiers will be used. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

The following steps have been taken to detect human face:

- Load the BGR image.
- Transform the image from BGR to Grayscale because it is easy to detect faces in the grayscale.
- Find faces in the image

Dog image detection

In this step, a model will be built to detect whether the supplied image contains dogs.

VGG-16 model, along with weights that have been trained on ImageNet will be used.

The following steps have been taken to detect dog images:

Load the image as input
Resize the image
Get the index corresponding to the ImageNet class that is predicted by the pre-trained VGG-16 model. The output should always be an integer between 0 and 999, inclusive.
If the index is between 151 and 268, the image contains dogs. (According to the dictionary.)

Dog breed prediction

CNN From Scratch
In this step, a CNN model will be created from scratch to estimate the dog breed from the supplied image.

The following steps have been taken to build the model:

- Create a loader to resize, horizontal and vertical flip and random rotation to augment the dog image dataset and preprocess the data.
- Create a CNN to classify dog breed.
- Specify loss function and optimizer.
- Train the data and save the final model parameters.
- Test the model using the testing data.

CNN using Transfer Learning

In this step, I use transfer learning to create a CNN that can identify dog breed from images. VGG16 has been picked as the pre-trained model since it is a well known good performing model and has been used in the dog image detection step.

The following steps have been taken to build the model:

- Use the data loader created in the previous step to load and preprocess the data.
- Initialise a pre-trained VGG16 model as the transfer learning model.
- Replace the upper layers of the network with a custom classifier then set the number of outputs to the number of dog breed classes that is set from the above model.
- Specify loss function and optimizer.
- Train the data and save the final model parameters.
- Test the model using the testing data.

Create prediction app

In this step, the human face detector, dog image detector and the CNN model have been integrated to an app which can detect the supplied image is a human or dog and provide the predicted dog breed.

The following steps have been taken to build the app:

- Check if the image is human by using the human face detector.
- Check if the image contains a dog by using the dog detector.
- Predict the dog breed from the image by using CNN using Transfer Learning with better performance results.

Benchmark

There is an article published on 2018 http://noiselab.ucsd.edu/ECE228_2018/Reports/Report17.pdf for some results of the existing models.

Table 2. Results out of different pre-trained model

Model	test accuracy	reference
VGG-19 and Resnet-50	82.30%	[2]
VGG16BN	76.53%	[3]
VGG16	40.23%	[3]
ResNet50	85.50%	[4]
ResNet50	84.00%	[5]
Inception v3	74%	[5]

It states that VGG16 with batch normalization has a test accuracy of 76.53%. Since the VGG16 with batch normalization is exactly the same per-trained model I am going to use for Transfer Learning, this number will be the comparison benchmark result.

Methodology

Data Preprocessing

There are 3 problems that we are solving in this project. Each of them have some level of data preprocessing.

Human face detection

Load the images and convert it to Grayscale using *cvtColor()* for OpenCV. The Grayscale images will be used to detect human faces in the later steps.

```
# load color (BGR) image
img = cv2.imread(human_files[0])
# convert BGR image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Dog image detection

Dog images are provided in different sizes so in this step I add a preprocess step to resize the images to 224x224.

```
preprocess = transforms.Compose([
        transforms.Resize((224,224)),
        transforms.ToTensor()])
img = Image.open(img_path)
img = preprocess(img)
img = img.unsqueeze(0)
```

Dog breed prediction

In this step, the dog images will be used to train, validate and test to create the CNN model. Since the data has already be splitted to train/validate/test folders with dog breed names as the folder names. There is no need to do the file splitting.

The preprocess steps are taken to augment the dataset:

- Resize the images to 224x224 for all images.
- Random horizontal/vertical flips for training and validation images.
- Random rotate the images for all images.

```
train_preprocess = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.RandomRotation(20),
    transforms.ToTensor(),
```

```
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])])

test_preprocess= transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomRotation(20),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])])

data_path = '/data/dog_images/'
train_data = datasets.ImageFolder(data_path + 'train/', transform=train_preprocess)
test_data = datasets.ImageFolder(data_path + 'test/', transform=test_preprocess)
valid_data = datasets.ImageFolder(data_path + 'valid/', transform=test_preprocess)
```

Implementation

Human face detection

A face_detector() function has been created to return True/False whether there are any faces found in the image.

```
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
```

Load the human face image and dog images then call face_detector() to test the model.

```
human_files_short = human_files[:100]
dog_files_short = dog_files[:100]

valid_count_human_files = 0
valid_count_dog_files = 0
for human_file in human_files_short:
    if face_detector(human_file):
        valid_count_human_files += 1

for dog_file in dog_files_short:
    if face_detector(dog_file):
        valid_count_dog_files += 1

print("Human files face detection percentage is {}%".format(valid_count_human_files))
print("Dog files face detection percentage is {}%".format(valid_count_dog_files
```

The testing results are:

Human files face detection percentage is 98%
Dog files face detection percentage is 17%

Dog image detection

In this step, I used a pre-trained VGG-16 model along with weights that 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.

A function *VGG16_predict()* has been created which can return the image class index.

```
import torch
import torchvision.models as models
from PIL import Image, ImageFile
import torchvision.transforms as transforms

ImageFile.LOAD_TRUNCATED_IMAGES = True

# define VGG16 model
VGG16 = models.vgg16(pretrained=True)

# check if CUDA is available
use_cuda = torch.cuda.is_available()

# move model to GPU if CUDA is available
if use_cuda:
    VGG16 = VGG16.cuda()

def VGG16_predict(model, img_path):
    '''
    Use pre-trained VGG-16 model to obtain index corresponding to
    predicted ImageNet class for image at specified path

    Args:
        img_path: path to an image

    Returns:
        Index corresponding to VGG-16 model's prediction
    '''

    ## Load and pre-process an image from the given img_path
    ## Return the *index* of the predicted class for that image
    preprocess = transforms.Compose([
```

```
        transforms.Resize((224,224)),
        transforms.ToTensor()])
    img = Image.open(img_path)
    img = preprocess(img)
    img = img.unsqueeze(0)

    if use_cuda:
        img = img.cuda()

    predictions = model(img)
    _, index = predictions.max(1)

    return index.item() # predicted class index
```

Then a *dog_detector()* function has been created to call *VGG16_predict()* and return True/False based on if the index value is between 151 and 268 - the dog classes from the ImageNet dictionary.

```
def dog_detector(model, img_path):
    index = VGG16_predict(model, img_path)
    return 151 <= index <= 268 # true/false
```

Test the result by using the human and dog files datasets.

```
### Test the performance of the dog_detector function
### on the images in human_files_short and dog_files_short.
valid_count_human_files = 0
valid_count_dog_files = 0
for human_file in human_files_short:
    if dog_detector(VGG16, human_file):
        valid_count_human_files += 1

for dog_file in dog_files_short:
    if dog_detector(VGG16, dog_file):
        valid_count_dog_files += 1

print("Human files face detection percentage is {}%".format(valid_count_human_files))
print("Dog files face detection percentage is {}%".format(valid_count_dog_files))
```

The results I have got here are:
Human files face detection percentage is 0%
Dog files face detection percentage is 93%

Dog breed prediction model

Build the CNN model from scratch.

Load the data from the preprocessed dataset using torch.utils.data.DataLoader() with different batch sizes.

```
train_loader= torch.utils.data.DataLoader(train_data, batch_size=32,
num_workers=0, shuffle=True)
valid_loader = torch.utils.data.DataLoader(valid_data, batch_size=16,
num_workers=0, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_data, batch_size=16,
num_workers=0, shuffle=True)

loaders_scratch = {'train' : train_loader, 'valid': valid_loader, 'test': test_loader}
```

I have created 3 convolutional layers with padding 1 to keep the output dimension the same as input for the initial network. Then a fully connected layer with a flatten 1D feature vector (28x28x64) is used for generating the 1024 output. Second fully connected layer is used to generate the 133 dog breed classes output.

In the forward function, the reLU activation layers are used combined with MaxPool2d to reduce the input size.

The MaxPool2d layer with 2x2 dimension and 2 strides is used to reduce the number of input sizes to improve the training performance. The (2, 2) parameter values are chosen from most common use cases and the performance is fine.

Apply the dropout layer to prevent overfitting and increase the performance. 0.25 is used since the recommendation is between 0.2 and 0.4.

Apply the batch normalisation to increase the training speed.

Below is the print out of the network.

```
Net(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=50176, out_features=1024, bias=True)
  (fc2): Linear(in_features=1024, out_features=133, bias=True)
  (dropout): Dropout(p=0.25)
  (batch_norm): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
```

Then specify the loss function using CrossEntropyLoss() and optimizer using SGD()

```
### select loss function
criterion_scratch = nn.CrossEntropyLoss()

### select optimizer
optimizer_scratch = optim.SGD(model_scratch.parameters(), lr=0.01)
```

After 15 epochs training, the test result has 14% accuracy. I then increased the number of epochs to 50 and got an increased accuracy of 17%. Given the validation was not decreased since epoch 34. There won't be much improvement with more epochs.

Build an app to integrate the 3 models

I created a script to integrate the 3 models which can handle a supplied image path and return the human face/dog detection result and provide the predicted dog breed.

Create the predict_breed() with the data and model built above.

```
data = {'train' : train_data, 'valid': valid_data, 'test': test_data}
# list of class names by index, i.e. a name can be accessed like class_names[0]
class_names = [item[4:].replace("_", " ") for item in data['train'].classes]

def predict_breed(img_path):
    # load the image and return the predicted breed
    preprocess = transforms.Compose([
        transforms.Resize((224,224)),
        transforms.ToTensor()])
    img = Image.open(img_path)
    img = preprocess(img)
    img = img.unsqueeze(0)

    if use_cuda:
        img = img.cuda()

    predictions = model(img)
    _, index = predictions.max(1)

    return class_names[index.item()]
```

Create the run_app() for the process.

```
def run_app(img_path):
    ## handle cases for a human face, dog, and neither
    def show_image(img_path, title="Title"):
        image = Image.open(img_path)
```

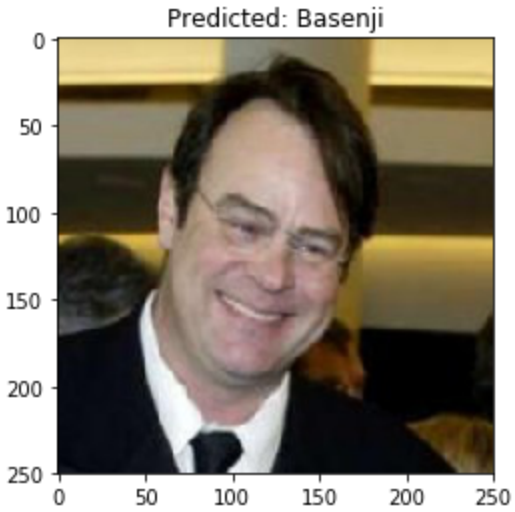
```
        plt.title(title)
        plt.imshow(image)
        plt.show()
    # check human faces:
    if (face_detector(img_path)):
        predicted_breed = predict_breed(img_path)
        show_image(img_path, title="Predicted: {}".format(predicted_breed) )

        print("This is a human image. Predicted breed is {}".format(predicted_breed))

    # check dogs:
    elif dog_detector(VGG16, img_path):
        predicted_breed = predict_breed(img_path)
        show_image(img_path, title="Predicted: {}".format(predicted_breed) )

        print("This is a dog image from path {}. Predicted breed is {}".format(img_path,
predicted_breed))
    # not humans and not dogs
    else:
        print("This is neither dog nor human image.")
        show_image(img_path, title="...")
```

This returns the detector and dog breed prediction result something like:



This is a human image. Predicted breed is Basenji

Refinement

The human face detector and dog image detector are using pre-trained models and achieved high accuracy. In this project, I didn't try to improve their performance.

The CNN model built from scratch has a low accuracy which was not satisfactory for this problem. I have used the transfer learning approach to build another CNN model.

In this case, I reused the data loader with the preprocessed dataset from the CNN model from scratch.

```
loaders_transfer = loaders_scratch
```

In this architecture, I reuse as much as possible from the "from scratch" CNN model above.

The VGG16 model has been created and set the "requires_grad" value to "False" so they are set to trainable by default.

Replace the upper layers of the network with a custom classifier then set the number of outputs to the number of dog breed classes that is set from the above model. In this way, only the custom classifier layer is trained.

```
import torchvision.models as models
import torch.nn as nn

## Specify model architecture
model_transfer = models.vgg16(pretrained=True)

for param in model_transfer.parameters():
    param.requires_grad = False

n_inputs = model_transfer.classifier[6].in_features
model_transfer.classifier[6] = nn.Linear(n_inputs, number_of_dog_breed_classes, bias=True)

if use_cuda:
    model_transfer = model_transfer.cuda()
```

The model architecture now looked as below:

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

```
  (6): ReLU(inplace)
  (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (8): ReLU(inplace)
  (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (11): ReLU(inplace)
  (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (13): ReLU(inplace)
  (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (15): ReLU(inplace)
  (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (18): ReLU(inplace)
  (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (20): ReLU(inplace)
  (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (22): ReLU(inplace)
  (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (25): ReLU(inplace)
  (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (27): ReLU(inplace)
  (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (29): ReLU(inplace)
  (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
 )
 (classifier): Sequential(
  (0): Linear(in_features=25088, out_features=4096, bias=True)
  (1): ReLU(inplace)
  (2): Dropout(p=0.5)
  (3): Linear(in_features=4096, out_features=4096, bias=True)
  (4): ReLU(inplace)
  (5): Dropout(p=0.5)
  (6): Linear(in_features=4096, out_features=133, bias=True)
 )
)
```

After 15 epochs training, it achieved the test result of 79% accuracy. Since it is better than the benchmark result of 76.53%, I didn't try to do more epochs.

Results

Model Evaluation and Validation

Human face detector

The testing results using pre-trained OpenCV are:

Human files face detection percentage is 98%
Dog files face detection percentage is 17%

The accuracy rate 98% for human face detection is a very high number. Although there is a relatively high incorrect rate of 17% that the detector finds dog images as human faces, it will not affect the final goal of dog breed prediction.

Dog image detector

The testing results using pre-trained VGG16 are:

Human files face detection percentage is 0%
Dog files face detection percentage is 95%

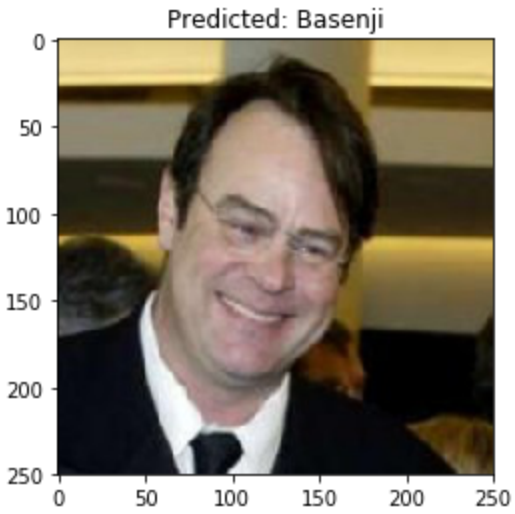
The accuracy of this model is quite good with 0% of human images detected as dogs and 95% of dogs images detected as dogs.

Dog breed predictor

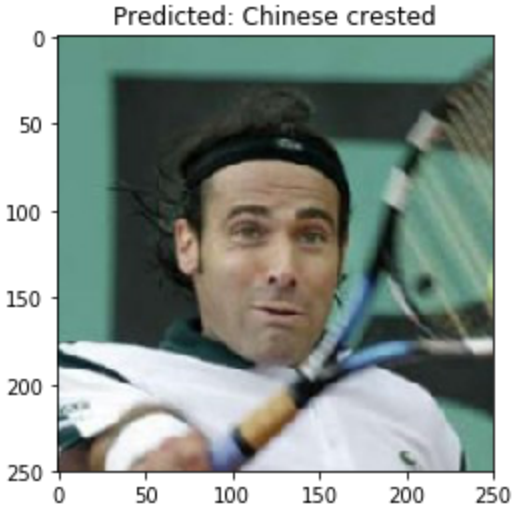
The model with transfer learning has achieved 79% after 15 epochs training. It is already a good result with the validation loss not showing overfitting issues. It is believed that more epochs training might increase the performance.

Integrated App

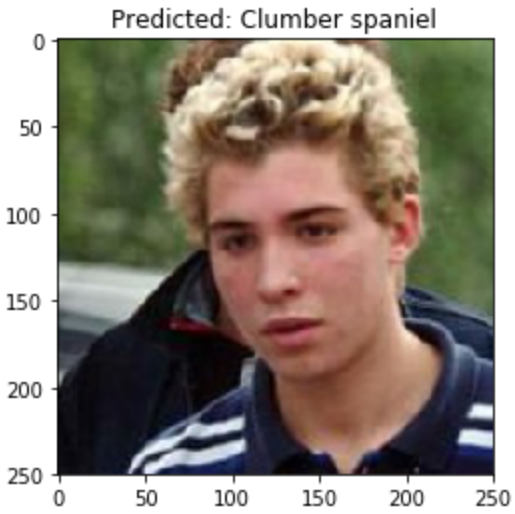
The app which integrated the models have tested results:



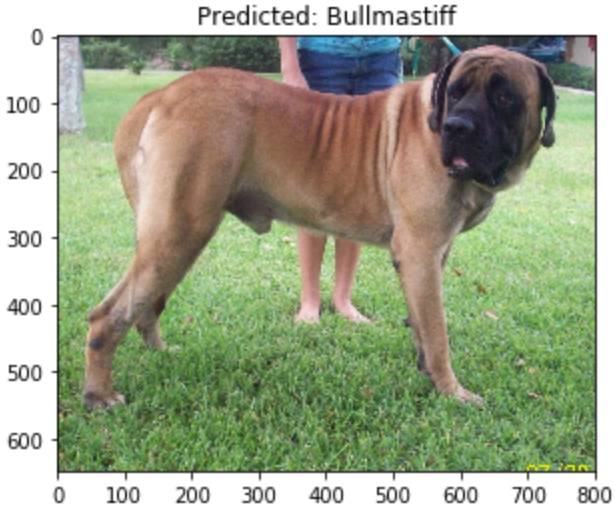
This is a human image. Predicted breed is Basenji



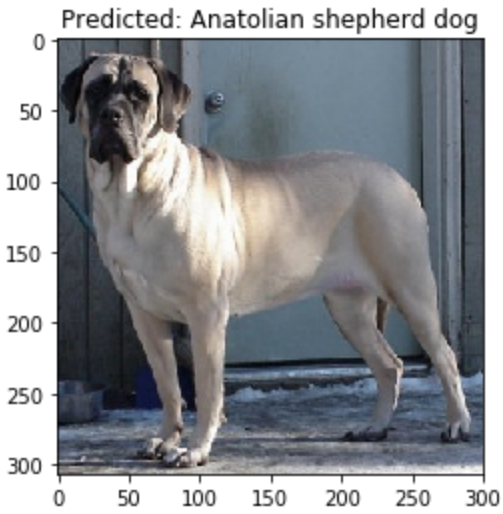
This is a human image. Predicted breed is Chinese crested



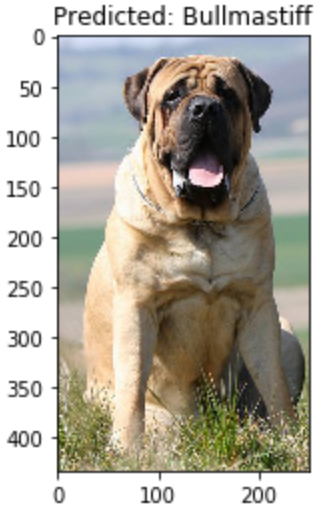
This is a human image. Predicted breed is Clumber spaniel



This is a dog image from path /data/dog_images/train/103.Mastiff/Mastiff_06833.jpg. Predicted breed is Bullmastiff



This is a dog image from path /data/dog_images/train/103.Mastiff/Mastiff_06826.jpg. Predicted breed is Anatolian shepherd dog



This is a dog image from path /data/dog_images/train/103.Mastiff/Mastiff_06871.jpg. Predicted breed is Bullmastiff

Justification

The final dog breed prediction model with transfer learning has achieved 79% which is better than the benchmark result of 76.53% using the same model (VGG16 with batch normalization).

There is still room to improve. Here are a few things that can be done to improve the performance of the model.

- VGG16 is an easy to use and fast training model but it is not the most accurate. There are some models with better accuracy such as VGG19, ResNet152, Densenet161 and InceptionV3.
- In this project, I only trained the model for 15 epochs(to save time and resources). Increasing it to 40-50 epochs should improve the accuracy of the model.
- The dog breed images are limited. If I can get more images for each dog breed, it will increase the accuracy of the model.
- Different numbers of CNN layers and batch size can be tried to find out the best accuracy architecture.

The app for integrated 3 models performed as expected. All the human face images are detected as human and dog images are detected as dogs.

There is no easy way to justify if the human images have the correct predicted resembling dog breed.

The dog breed prediction result is not good as expected. 2 “mastiff” dogs have been predicted as “bullmastiff”. 1 “mastiff” was predicted as an “anatolian shepherd”. If “mastiff” predicted as “bullmastiff” are considered the correct answer, the accuracy rate is 66.7% which is lower than the test results.