Capstone Project Report
Omar Alejandro Martinez Garcia

CNN Project: Dog Breed Classification

Project Overview

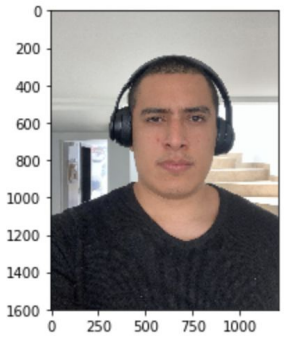
Computer vision is the field of study that focuses on how to train computers to understand and interpret visual information such as images or videos. The resulting models have shown to be really useful for different applications such as self-driving cars, facial recognition, defect detection on production lines, and more. However, the training of these models usually requires massive amounts of data and the performance of the developed algorithms varies between different cases. The projects within the field of computer vision are often challenging as the benchmark for comparison is human vision, which is incredibly successful in performing these types of tasks.

This project goal is to create a model that receives images as inputs and identifies the presence of a human or a dog and the canine breed they resemble or belong.

Problem Statement

The dog breed classification problem focuses on solving two main problems. First, the model is trained to recognize if the input image contains a human or a dog, and second, it identifies the breed that these resemble. It is a supervised multi-class classification problem that predicts among 133 types of dog breeds. The following is the expected output:

```
Hello, human!
```




```
You look like a ...
 Havanese!
```

Metrics:

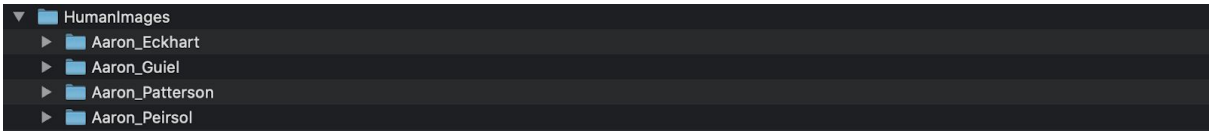
For this classification problem, the cross-entropy loss will be used for training the models and the accuracy is used as the evaluation metric. This metric is useful for this classification task, as we want to know how many predictions are done correctly over the dataset. It evaluates if the models are identifying the presence of humans and dogs and if the latter are correctly classified within the corresponding breeds.

$$\text{Accuracy} = \frac{correct\ predictions}{all\ predictions} = \frac{TP + TN}{TP + TN + FP + FN}$$

Analysis

The dataset is provided by Udacity and is composed of images of humans and dogs. Both of these are not balanced as the number of images per dog breed and individuals vary.

- There are 13233 total human images that correspond to 5750 different people and all have a size of 250x250 pixels. As we can see in the following examples, the images have a different background, lighting, angles, expressions, and poses.



- There are 8351 total dog images divided into train, test, and valid directories that contain different samples of images for the 133 dog breeds that will be classified. These images vary in size, background, zoom, angle, pose, and lighting.
    - Train folder: 6680 images
    - Test: 836 images
    - Valid: 835 images

Methodology

The implementation of the project consists of the following 7 major steps in order to achieve the desired outcome.

1. Import Datasets

In order to import the human and dogs datasets, it is necessary to download and unzip the files at the project's home directory in the locations /dog_images and /lfw. The following code works for loading the filenames for the datasets.

```
import numpy as np
from glob import glob

# load filenames for human and dog images
human_files = np.array(glob("/data/lfw/*/*"))
dog_files = np.array(glob("/data/dog_images/*/*/*"))

# print number of images in each dataset
print('There are %d total human images.' % len(human_files))
print('There are %d total dog images.' % len(dog_files))
```

```
There are 13233 total human images.
There are 8351 total dog images.
```

2. Face Detector Model

Used a pre-trained face detector from OpenCV's implementation of the Haar feature-based cascade classifier to detect human faces in images using the following code:

Pre-processing

The detectMultiScale function takes grayscale images so it is necessary that the image is converted to grayscale before passing it through the model.

Model

```
# extract pre-trained face detector
face_cascade = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_alt.xml')
```

```
# 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
```

Results

The performance of the model was assessed and it was identified that:

- In the first 100 images in human_files, the model detected that 98% corresponded to a human face
- In the first 100 images in dog_files, the model detected that 17% corresponded to a human face

3. Dog Detector Model

The pre-trained VGG-16 model on ImageNet was used as a detector that classifies whether there is a dog on the input images. The model was imported using Pytorch:

```
import torch
import torchvision.models as models

# 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()
```

Pre-processing:

As it is explained in the PyTorch documentation the images should be appropriately pre-processed. For this reason, a transformation pipeline was created in which images were resized to 224 x 224 pixels and normalized with a mean of (0.485, 0.456, 0.406) and a standard deviation of (0.229, 0.224, 0.225).

```
pipeline = transforms.Compose([transforms.Resize((224,224)),
                               transforms.ToTensor(),
                               transforms.Normalize((0.485, 0.456, 0.406),
                                                    (0.229, 0.224, 0.225))])
```

Additionally, as the model requires a 4-dimensional input tensor of shape (n_samples, rows, columns, channels), the result of the pipeline transformation is “unsqueezed” to compensate for the first missing dimension.

```
img = pipeline(Image.open(img_path)).unsqueeze(0)
```

As the output of the pre-trained VGG-16 model is associated with a dictionary of 1000 different categories and only those keys between 151-268 include different dog breeds. It was decided that the dog detector would only determine the presence of a dog, only if the output of the model correspond to any of the dictionary keys within the following range [151, 268].

Results

In the end, the VGG-16 model identified that 0% of the first 100 images on human_files have a dog and that 100% of the first 100 images on dog_files have detected a dog face.

4. CNN to classify dog breeds from scratch

This is the **benchmark** model and it should perform better than random guessing in the dog breed classification task, which in this case corresponds to an accuracy of less than 1%. In fact, it should have at least an accuracy of 10%.

Pre-processing:

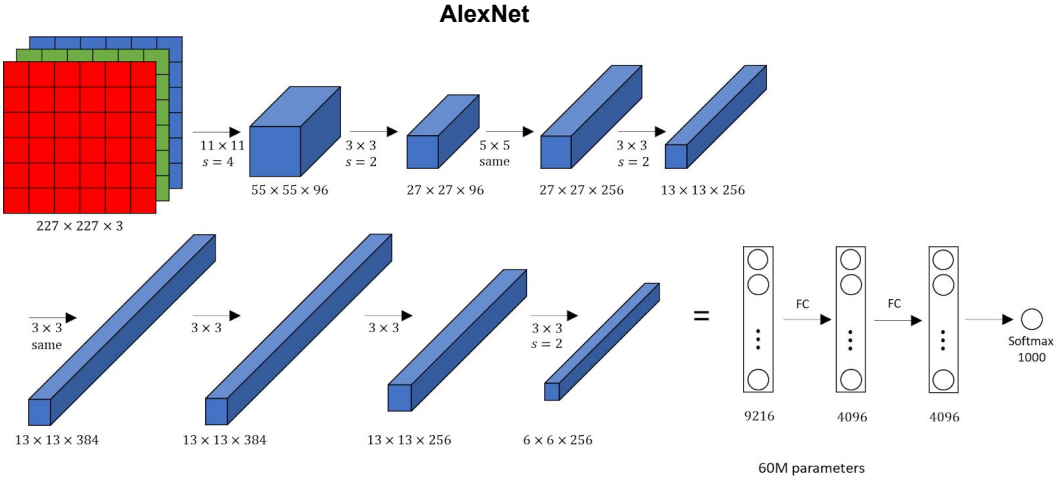
As a simplified version of the Alexnet architecture was implemented the following dictionary was created with the transformation pipelines that were necessary for the training, validation, and test set.

```
# Dictionary with transformation pipelines for train and test(used also in validation)
transformations = {'train': transforms.Compose([transforms.Resize((227,227)),
                                                transforms.RandomRotation(20),
                                                transforms.RandomHorizontalFlip(),
                                                transforms.RandomVerticalFlip(),
                                                transforms.ToTensor(),
                                                transforms.Normalize((0.485, 0.456, 0.406),
                                                                     (0.229, 0.224, 0.225))]),

                   'test': transforms.Compose([transforms.Resize((227,227)),
                                               transforms.ToTensor(),
                                               transforms.Normalize((0.485, 0.456, 0.406),
                                                                    (0.229, 0.224, 0.225))])}
```

- Training dataset: Images were resized to 227 x 227, and random rotations of 20 degrees, random horizontal flips, and random vertical flips were performed to augment the data and help the model to generalize better on the input images
- Val and Test datasets: The same transformations were applied to both sets of images and they were only resized to 227 x 227 without any other type of transformations or augmentations.

Architecture:

In order to achieve at least 10% accuracy on the test dataset, AlexNet (A milestone for CNN Image Classification) architecture was select for the model. During the training of the full Alexnet architecture, it was clear that the model was suffering from low bias and high variance problem. It was probable that the complexity of the network and the limited amount of data were making the model overfit the training set.



For this reason, the final architecture was simplified as the following:

```
import torch.nn as nn
import torch.nn.functional as F

# define the CNN architecture
class Net(nn.Module):
    ### TODO: choose an architecture, and complete the class
    def __init__(self):
        super(Net, self).__init__()
        ## Define layers of a CNN

        self.conv1 = nn.Conv2d(3, 96, 11, stride=4, padding=0)
        self.conv2 = nn.Conv2d(96, 256, 5, stride=1, padding=2)
        self.conv3 = nn.Conv2d(256, 384, 3, stride=1, padding=1)
        self.conv4 = nn.Conv2d(384, 256, 3, stride=1, padding=1)

        self.fc1 = nn.Linear(256 * 6 * 6, 4096)
        self.fc2 = nn.Linear(4096, 133)

        self.pool = nn.MaxPool2d((3,3), stride=2)
        self.dropout = nn.Dropout(0.2)

    def forward(self, x):
        ## Define forward behavior
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = self.pool(x)

        # flattening x
        x = x.view(-1, 256 * 6 * 6)

        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)

        return x

#-#-# You so NOT have to modify the code below this line. #-#-#

# instantiate the CNN
model_scratch = Net()

# move tensors to GPU if CUDA is available
if use_cuda:
    model_scratch.cuda()
```

- A first convolutional layer that applies 96 filters of size 11 x 11 and a stride of 4 to reduce dimensions by a rough factor of 4.
- A max-pooling layer with a filter of size 3 x 3 and a stride of 2
- A **same** convolutional layer with 256 filters of size 5 x 5 and padding of 2
- A max-pooling layer with a filter of size 3 x 3 and a stride of 2
- A **same** convolutional layer with 384 filters of size 3 x 3 and padding of 1
- A **same** convolutional layer with 256 filters of sixe 3 x 3 and padding of 1
- A max-pooling layer with a filter of size 3 x 3 and a stride of 2
- Two fully connected layers with a dropout layer in between and an output of 133 (For the classification of the 133 different breeds)

In summary, one **same** convolutional layer with 384 filters of size 3 x 3 and padding of 1 and a fully connected layer were removed. Additionally, a dropout layer with p=0.2 was included.

Results

After 25 epochs of training with mini-batches of size 32 and a learning rate of 0.05, the model achieved a 12% accuracy on the test set, which surpassed the expectation of 10%.

5. CNN to classify Dog Breeds Using Transfer Learning

The ResNet50 model was chosen for this task as it has shown outstanding performance on image classification problems thanks to the shortcut connections that enable a deeper network that performs better on training.

Preprocessing

As the ResNet model was used for the transfer learning classification task, it was necessary to redesign the previous transformation pipelines used for the AlexNet. In this case, everything remained the same and the only modification was a 224 x 224 resizing, instead of a 227 x 227.

```
transformations = {'train': transforms.Compose([transforms.Resize((224,224)),
                                                transforms.RandomRotation(20),
                                                transforms.RandomHorizontalFlip(),
                                                transforms.RandomVerticalFlip(),
                                                transforms.ToTensor(),
                                                transforms.Normalize((0.485, 0.456, 0.406),
                                                            (0.229, 0.224, 0.225))]),

                   'test': transforms.Compose([transforms.Resize((224,224)),
                                               transforms.ToTensor(),
                                               transforms.Normalize((0.485, 0.456, 0.406),
                                                                    (0.229, 0.224, 0.225))])}
```

Architecture

The general architecture of the ResNet50 remained the same and the parameter values were frozen except for the last layer which was replaced for a fully connected layer with an output of 133.

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

model_transfer = models.resnet50(pretrained=True)

#Freeze parameters of the pre-trained model
for param in model_transfer.parameters():
    param.requires_grad = False

#New Layer
model_transfer.fc = nn.Linear(2048, 133, bias=True)

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

Results

After 25 epochs of training with mini-batches of size 32 and a learning rate of 0.01, the model achieved an 83% accuracy on the test set, which surpassed the expectation of 60%. A great performance.

6. Integration of Face Detector, Dog Detector, and ResNet50 as a Final Model

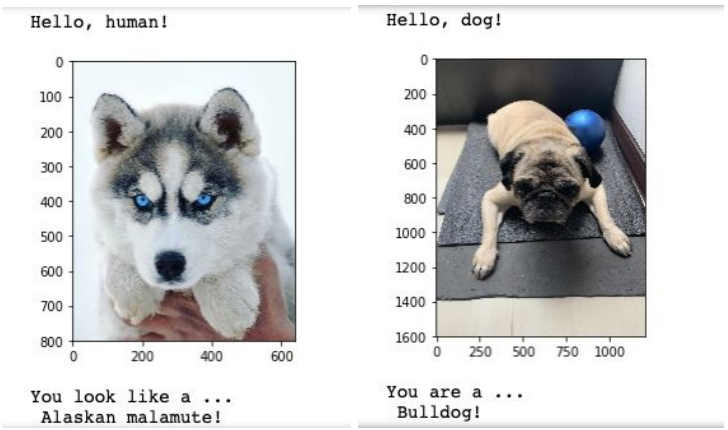
The algorithm works as follows:

```
def run_app(img_path):

    img = Image.open(img_path)
    plt.imshow(img)
    pred = predict_breed_transfer(img_path, class_names)

    if face_detector(img_path):
        print("\nHello, human!")
        plt.show()
        print("You look like a ...\n {}!\n ".format(pred))

    elif dog_detector(img_path):
        print("\nHello, dog!")
        plt.show()
        print("You are a ...\n {}!\n".format(pred))

    else:
        print("No human or dog identified...")
        plt.show()
```

- If the face detector detects the presence of a human face on the input image, it will print the message “Hello, Human” and predict the dog breed that the human resembles the most.
- In case it doesn’t identify a face, the input image is passed to the dog detector, in which the image can be classified as a dog and the predicted breed would be outputted.
- In case non of the detectors detects the presence of a human face or a dog, the algorithm will simply print “No human or dog identified”

7. Results Final Model

The model performed better than expected on 6 images (3 humans and 3 dogs) with different attributes. However, it still not perfect as it classified a dog like a human and a pug breed as a bulldog. It is not surprising as the face detectors previously predicted human faces on the dog images and the similarity between pug breeds and bulldogs is high.



Results Summary

Face Detector

The performance of the model was assessed and it was identified that:

- In the first 100 images in human_files, the model detected that 98% corresponded to a human face
- In the first 100 images in dog_files, the model detected that 17% corresponded to a human face

It would be interesting to research other face detection pre-trained models and compare their performance. It is possible that there is a model that could perform better on the dog dataset.

Dog Detector

The performance of the Dog Detector is outstanding as the VGG-16 model identified that 0% of the first 100 images on human_files have a dog and that 100% of the first 100 images on dog_files have detected a dog face. It would be advised to check if this performance maintains when evaluating a larger part of the datasets.

CNN to classify Dog Breeds from Scratch

After 25 epochs of training with mini-batches of size 32 and a learning rate of 0.05, the simplified AlexNet based model achieved a 12% accuracy on the test set (104/836), which surpassed the expectation of 10%. This a good performance taking into account the complexity of the network, the training time, and the amount of available data.

CNN to classify Dog Breeds Using Transfer Learning

After 25 epochs of training with mini-batches of size 32 and a learning rate of 0.01, the ResNet50 model achieved an 83% accuracy on the test set (696/836 images), which surpassed the expectation of 60%. A great performance.

Final Model

The model performed better than expected on 6 images (3 humans and 3 dogs) with different attributes. However, it still not perfect as it classified a dog like a human and a pug breed as a bulldog. It is not surprising as the face detectors previously predicted human faces on the dog images and the similarity between pug breeds and bulldogs is high.

Conclusion

The results are outstanding and the usefulness of pre-trained algorithms is clear. However, It would be interesting to perform the following proposals and evaluate if there is an increase in performance:

- Gather more data (images) and perform more augmentation for training
- Unfreeze more parameters of the model used for transfer learning (ResNet50) so it would perform better on this task
- Train for more epochs
- Used other models for Face Detection
- Use other models for transfer learning such as VGGs, Inception, GoogLeNet, and more

References

https://pytorch.org/docs/stable/torchvision/models.html
https://www.programmersought.com/article/23174986697/
http://datahacker.rs/deep-learning-alexnet-architecture/
https://github.com/udacity/deep-learning-v2-pytorch/blob/master/project-dog-classification/
https://towardsdatascience.com/computer-vision-and-why-it-is-so-difficult-1ed55e0d65be
https://gombru.github.io/2018/05/23/cross_entropy_loss/