UDACITY CAPSTONE REPORT

DOG BREED IDENTIFICATION

CNN (Convolutional Neural Networks) Project

Project Overview:

The Dog Breed Identification problem is very popular problem on Kaggle. In this project, the given image is of dog or human. If it is of dog, we identify the breed of the dog while if the image is of human, we identify the resembling dog breed. The given problem is multi class classification problem which solved using supervised learning.

The project consists of following steps:

Step 0: Import Datasets

Step 1: Detect Humans

Step 2: Detect Dogs

Step 3: Create a CNN to classify Dog Breeds (from Scratch)

Step 4: Create a CNN to classify Dog Breeds (using Transfer Learning)

Step 5: Writing Algorithm

Step 6 : Testing the Algorithm.

Problem Statement:

This project uses Convolutional Neural Network with Transfer Learning to identify the dog breeds. Transfer learning has advantage that it can decrease time to develop and train a model by reusing the modules of already developed models. Therefore, the model training process speeds up.

The algorithm must achieve three objectives:

i) Dog Detector: given an image of a dog, algorithm will identify an estimate of the canine's breed

ii) Human Detector: If supplied an image of a human; the code will identify the resembling dog breed.

iii) If the image supplied is identified as neither human nor dog, then the algorithm would output “Invalid Image”.

Metrics:

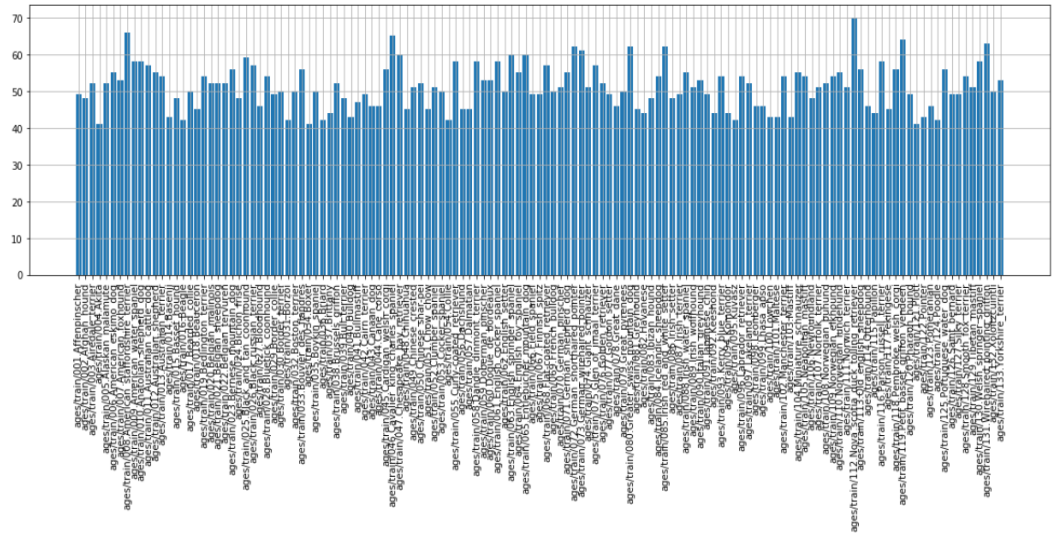
Multi class log loss is used to evaluate the model. It takes in account the uncertainty of prediction based of variation from the actual label. The non-uniformity of dataset is the reason for not taking accuracy as indicator. The data is split into train, test and validate datasets. Performance of model is checked by using testing dataset after training the model through train dataset.

Data Exploration and Visualization:

The dataset of the project is provided by Udacity. To download the dataset, use following links:

For dog images Dataset: https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip

Dog image data has 8351 total images which includes 6680 images in train directory, 836 images in test directory and 835 images in valid directory. The data is not uniform. Some breed have more images than other.



Sample image from the dataset:

For Human images Dataset: https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip

The human image dataset contain total 13233 images. These are sorted by names of human in 5750 folders. All images are of same size i.e. 250x250. A sample image is shown:

Code Used:

```
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.
```

Benchmark

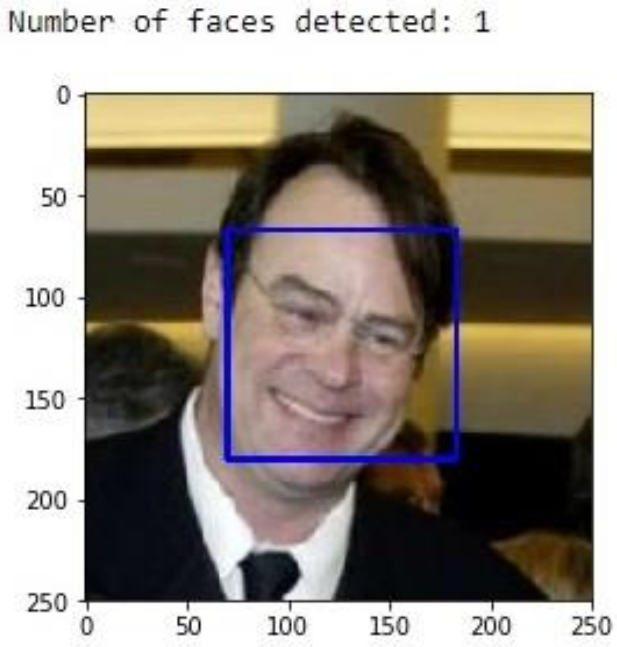
There are 3 benchmark algorithms in this project:

1) We have used OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in Images. OpenCV provide many pre-trained face detectors. We have download one of these detectors and stored it in haarcascades directory.

   The following code is used for implementation:

```
# Extract pre-trained face detector
face_cascade =
cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_al
t.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
```

   The face detector was tested on 100 human images out of which 98% were correctly classified as human. When we used face detector on 100 dog images, it mistakenly classified 17% dog images as humans.



2) We used pre-trained VGG-16 Model, along with weights that have been trained on ImageNet, a very popular dataset used for image classification and other vision tasks.
After downloading the data, we define VGG16_predict function as following:

```
def VGG16_predict(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
    '''

    ## TODO: Complete the function.
    ## Load and pre-process an image from the given img_path
    ## Return the *index* of the predicted class for that image

    image = Image.open(img_path).convert('RGB')

    normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],std=[0.229, 0.224, 0.225])

    transformations = transforms.Compose([transforms.Resize(size=(224, 224)),
                                          transforms.ToTensor(),
                                          normalize])

    transformed_image = transformations(image)[:3,:,:].unsqueeze(0)

    if use_cuda:
        new_image = transformed_image.cuda()

    out = VGG16(new_image)

    return torch.max(out,1)[1].item()
```

we then define a dog_detector function which returns true if a dog is detected in an image (and False if not).

```
### returns "True" if a dog is detected in the image stored at img_path
def dog_detector(img_path):
    ## TODO: Complete the function.

    predict_index = VGG16_predict(img_path)

    output = predict_index >=151 and predict_index <=268

    return output # true/false
```

Out of 100 dog images, all are correctly detected while out of 100 human images, 1 % have been wrongly classified as dogs.

3) CNN to Classify Dog Breeds (from Scartch):
   CNN is constructed to classify dog breed from given images.

```
# 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, 36, 3, padding=1)
        self.conv2 = nn.Conv2d(36, 64, 3, padding=1)
        self.conv3 = nn.Conv2d(64, 128, 3, padding=1)
        self.fc1 = nn.Linear(28*28*128, 512)
        self.fc2 = nn.Linear(512, 133)
        self.pool = nn.MaxPool2d(2, 2)
        self.dropout = nn.Dropout(0.25)
        self.batch_norm = nn.BatchNorm1d(512)

    def forward(self, x):

        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))

        x = x.view(-1, 28*28*128)

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

        return x
```

This model has 3 convolutional layers. The first layer have in_channels =3 and the final layer gives output size of 128. All the layers have kernel size of 3. ReLu function is also used here.

Data Preprocessing

All the images are resized to 224*224, then normalization is applied to all images (train, valid and test datasets). For the training data, Image augmentation is done to reduce overfitting.

```
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                  std=[0.229, 0.224, 0.225])

train_dataset = datasets.ImageFolder(train_path,  transforms.Compose([
            transforms.RandomResizedCrop(224),
            transforms.RandomHorizontalFlip(),
            transforms.RandomRotation(15),
            transforms.ToTensor(),
            normalize,
        ]))

validation_dataset = datasets.ImageFolder(validation_path,  transforms.Compose([
            transforms.Resize(size=(224,224)),
            transforms.ToTensor(),
            normalize,
        ]))

test_dataset = datasets.ImageFolder(test_path,  transforms.Compose([
             transforms.Resize(size=(224,224)),
            transforms.ToTensor(),
            normalize,
        ]))
```

Implementation

I have used CNN from scratch for implementation. The model has 3 convolutional layers. All layer have kernel size of 3 and stride 1. The first layer takes 224x224 image and final layer gives an output size of 128. ReLu activation function is used here. Pooling layer of (2,2) is used to reduce input size by 2. The two fully connected layers produces 133 dimensional output.

Refinement

The CNN from scratch model gives accuracy of 12%. To further improve accuracy, we use transfer learning. After specifying data loaders for training, validation and test datasets. We specify the model architecture.

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

## TODO: Specify model architecture

model_transfer = models.resnet101(pretrained=True)

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

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

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

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

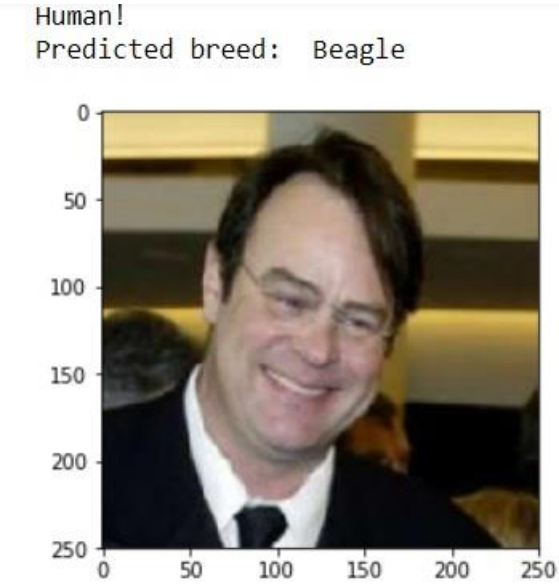
Then we specify Loss function and Optimizer:

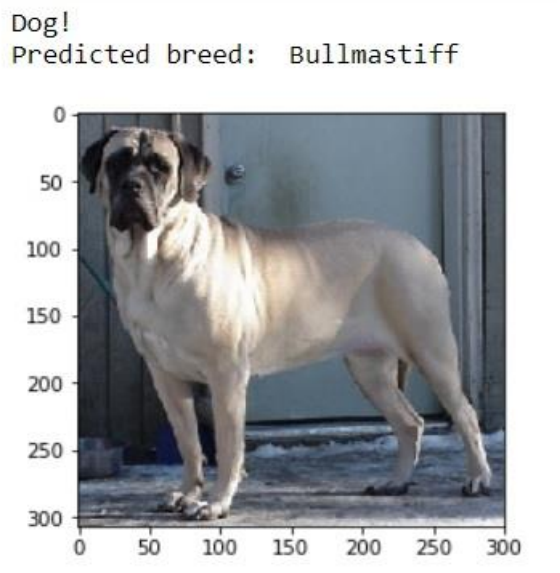
```
criterion_transfer = nn.CrossEntropyLoss()
optimizer_transfer = optim.SGD(model_transfer.fc.parameters(), lr=0.02)
```

We then, train, validate and test the model.

The CNN created from scratch have accuracy of 12% while CNN with transfer learning (Resnet101 architecture) which is pre-trained on ImageNet dataset, The model performed extremely well. With just 5 epochs, the model got 79% accuracy.

Sample outputs predicted using the model are shown:





Algorithm

The algorithm accepts a file path to an image and then determine if the image contains a human, dog or neither. Then,

- If a dog is detected in the image, return the predicted breed.
- If a human is detected in the image, return the resembling dog breed.
- If neither is detected in the image, return error message of “Invalid image”.

```
### TODO: Write your algorithm.
### Feel free to use as many code cells as needed.
def load_image(img_path):
    img = Image.open(img_path)
    plt.imshow(img)
    plt.show()

def run_app(img_path):
    ## handle cases for a human face, dog, and neither

    if face_detector(img_path):
        print ("Human!")
        predicted_breed = predict_breed_transfer(img_path)
        print("Predicted breed: ",predicted_breed)
        load_image(img_path)

    elif dog_detector(img_path):
        print ("Dog!")
        predicted_breed = predict_breed_transfer(img_path)
        print("Predicted breed: ",predicted_breed)
        load_image(img_path)

    else:
        print ("Invalid Image")
```

Model Evaluation and Validation

Using OpenCV’s implementation of Haar feature based cascade classifiers we made human face detector which detected 98% as human face in first 100 images of human face dataset and 17% of human faces were detected in first 100 images of dog dataset.

We used VGG16 model to make a Dog face detector which detected 100% of dog faces in first 100 images of dog dataset and 1% of dog faces in first 100 images of human dataset.

The CNN model created using transfer learning with ResNet101 architecture was trained for 5 epochs, and the final model produced an accuracy of 79% on test data. The model correctly predicted breeds for 668 images out of 836 total images. Accuracy on test data: 79% (668/836)

Justification

The model has performed better than my expectation. The CNN model from scratch had accuracy of only 12% while the refined transfer learning model obtained an accuracy of 79%.

Improvement

For improving the model, more training and testing data could be used. The current model uses only 133 breeds of dogs. Also by image augmentation we can avoid overfitting and thus improve the accuracy. In this project, ResNet101 architecture for feature extraction has been used. A different architecture may give better performance.

References:

1. Github repo of Project: https://github.com/udacity/deep-learning-v2-pytorch/tree/master/project-dog-classification
2. Resnet101: https://pytorch.org/docs/stable/_modules/torchvision/models/resnet.html#resnet101
3. Imagenet training in Pytorch: https://github.com/pytorch/examples/blob/97304e232807082c2e7b54c597615dc0ad8f6173/imagenet/main.py#L197-L198
4. Pytorch Documentation https://pytorch.org/docs/master/
5. CNN https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
6. Log Loss http://wiki.fast.ai/index.php/Log_Loss
7. Transfer Learning https://machinelearningmastery.com/transfer-learning-for-deep-learning/
8. ResNet: https://medium.com/analytics-vidhya/enhance-learning-by-transfering-resnet-architecture-into-big-transform-architecture-44603f537fcf