Capstone Report **Suryanshu**

Machine Learning Engineer Nanodegree

DOG BREED CLASSIFIER

Project Overview:

Dog breed identification problem is well known in the ML community. We can find it on Kaggle: *https://www.kaggle.com/c/dog-breed-identification/overview/description* It is intended to build a pipeline involving Convolutional Neural Networks (CNN) that can be used to process real-world, user-supplied images. Given an image of a dog, the algorithm will identify an estimate of its breed. If supplied with an image of a human, the code will identify the resembling dog breed.

Problem Statement

The problem is to identify a breed of dog if dog image is given as input, if supplied an image of a human, we have to identify the resembling dog breed. The idea is to build a pipeline that can process real world user supplied images and identify an estimate of the canine's breed.

Evaluation Metrics

As it's multiclass classification problem, using CrossEntropy Loss would be the best indicator . It's because takes into the account of uncertainty of prediction based on how much it varies from actual label.

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Binary Cross-Entropy / Log Loss

(IMPLEMENTATION) Specify Loss Function and Optimizer

Use the next code cell to specify a loss function and optimizer. Save the chosen loss function as `criterion_scratch`, and the optimizer as `optimizer_scratch` below.

```
import torch.optim as optim

### TODO: select loss function
criterion_scratch = nn.CrossEntropyLoss()

### TODO: select optimizer
optimizer_scratch = optim.Adam(model_scratch.parameters(), lr = 0.001)
```

Project Design

Step-1 Download dataset, unzip it and store it in the directory.

Step-2 Import necessary libraries, pre-process image and store it in train, test an validation dataset.

Step-3 Write function to detect human faces using OpenCV implementation of Haar feature based cascade classifiers.

Step-4 Write function to detect dogs using pre-trained VGG16 model.

Step-5 Create a CNN to classify Dog breeds from scratch. Train, validate and test the model.

Step-6 Create a CNN using transfer learning with VGG 16. Train and test the model.

Step-7 Write a function that combines Dog detector, face detector and Dog breed predictor model so that

- 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, provide output that indicates an error.

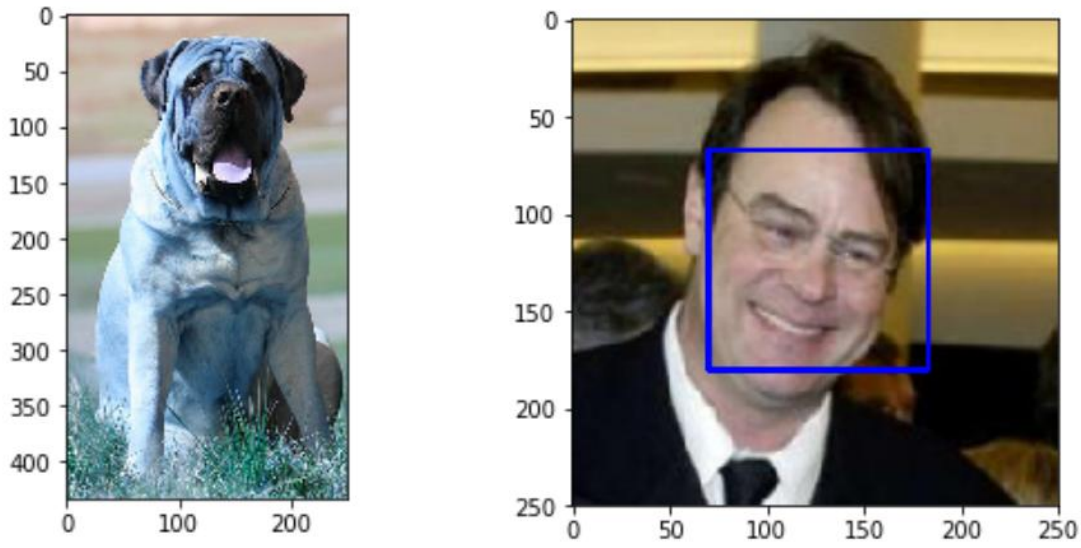
Datasets Exploration

The dataset for this project is provided by Udacity. The dataset has pictures of dogs and humans.

Dog images dataset: It has 8351 total images which are divided into train (6,680 images), test (836 images) and valid (835 images) directories. Each of this directory has 133 folders corresponding to dog breeds. Images have different background and different angles.

Human images dataset: The human dataset contains 13233 human images which are divided into folders by names of human (5750 folders). All images are of size 250x250. Images have different background and different angles.

Input for performing this classification is an Image. I am transforming images into 256*256 and then performing CenterCrop on Images to turn them into 224*224. I am normalizing images with mean values =[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]



Sample Images from Dataset

Algorithms and Techniques

I used Convolutional Neural Network(CNNs) to solve this problem. A **Convolutional Neural Network (ConvNet/CNN)** is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.

Initially, to detect human images, I used existing algorithm like OpenCV's implementation of Haar feature based cascade classifiers. Later, to detect dog-images I used a pretrained VGG16 model. Finally, after the image is identified as dog/human, I passed this image to my CNN which after processing through the layers defined predict the breed that matches most out of 133 breeds.

Benchmark Model

Using Transfer Learning with VGG-16 resulted in accuracy of around 80%. VGG16 Design was aimed at winning ILSVR Competition in 2014 where dataset contained many categories of images. It is considered to be one of the excellent vision model architecture till date. Most unique thing about VGG16 is that instead of having a large number of hyper-parameter they focused on having convolution layers of 3x3 filter with a stride 1 and always used same padding and maxpool layer of 2x2 filter of stride 2. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end it has 2 FC(fully connected layers) followed by a softmax for output. The 16 in VGG16 refers to it has 16 layers that have weights. This network is a pretty large network and it has about 138 million (approx) parameters.

Data Preprocessing

Input for performing this classification is an Image. I am transforming images into 256*256 and then performing CenterCrop on Images to turn them into 224*224. I am normalizing images with mean values =[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225] and further loading them into train, valid and test directories.

```
transforms = transforms.Compose([transforms.Resize(256),
                                 transforms.CenterCrop(224),
                                 transforms.ToTensor(),
                                 transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                                      std=[0.229, 0.224, 0.225])])

### TODO: Write data loaders for training, validation, and test sets
train_data = datasets.ImageFolder(data_dir + '/train', transform=transforms)
valid_data = datasets.ImageFolder(data_dir + '/valid', transform=transforms)
test_data = datasets.ImageFolder(data_dir + '/test', transform=transforms)
```

Implementation

Depth of layers for a RGB Image is 3, it is passed through three convolutional layers increasing depth from 3->16, 16->32, ,32->64. Image size is 224*224 after center cropping we did initially. After passing through each convolutional layer, ReLU activation and maxpooling layer (2,2) size is applied thus reducing dimensions by 2 after each pass. Finally output is flattened and is connected to fully connected layer which is connected to other fully connected that returns predictions(i.e 133 Dimensional output). A dropout of 0.25 is applied to avoid overfitting.

```python
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, 16, 3, padding=1)
        self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
        self.conv3 = nn.Conv2d(32, 64, 2, padding=1)
        self.conv3_bn = nn.BatchNorm2d(64)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(64 * 28 * 28, 500)
        self.fc1_bn = nn.BatchNorm1d(500)
        self.fc2 = nn.Linear(500, 133)
        self.dropout = nn.Dropout(0.25)

    def forward(self, x):
        ## Define forward behavior
        x = self.pool(F.elu(self.conv1(x)))
        x = self.pool(F.elu(self.conv2(x)))
        x = self.pool(self.conv3_bn(F.elu(self.conv3(x))))
        x = x.view(-1, 64 * 28 * 28)
        x = self.dropout(x)
        x = F.relu(self.fc1_bn(self.fc1(x)))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

Model and Evaluation

Human Face detector: The human face detector function was created using OpenCV's implementation of Haar feature based cascade classifiers. 98% of human faces were detected in first 100 images of human face dataset and 17% of human faces detected in first 100 images of dog dataset.

Dog Face detector: The dog detector function was created using pre-trained VGG16 model. 100% of dog faces were detected in first 100 images of dog dataset and 1% of dog faces detected in first 100 images of human dataset.

CNN made from scratch: The CNN model created from scratch was trained for 15 epochs, and it produced an accuracy of 11% on test data. The model correctly predicted breeds for 92 images out of 836 total images.

```
Test Accuracy: 11% (92/836)
```

CNN using transfer learning: The CNN model created using transfer learning with VGG-16 architecture was trained for 4 epochs, and the final model produced an accuracy of 79% on test data. The model correctly predicted breeds for 664 images out of 836 total images.

```
Test Accuracy: 79% (664/836)
```

Refinement

CNN created from scratch produced only an accuracy of 11% so we decided to build a model using transfer learning to predict the Dog breed. I used VGG 16 and removed last layer to map it to 133 dimensional output of our Dog breed and trained it for 4 epochs which resulted in a significant jump in accuracy to 79%.

Justification

We used CNNs as the pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

Model created using Transfer Learning is performing very well and is giving an accuracy of **79%** as compared to CNN created with scratch which yielded only an accuracy of **11%.**

Improvement

Better hyperparameters could have been chosen and 11% accuracy is very less on CNN made with scratch. Face detector could have been better. Image augmentation could have been added further to make model more robust to detect input image in any angle. Different architectures like Resnet 101 could be used for transfer learning to generate comparative analysis.