Capstone Project

Machine Learning Engineer Nanodegree

Bhadresh Savani

Dog Breed Classifier using CNN

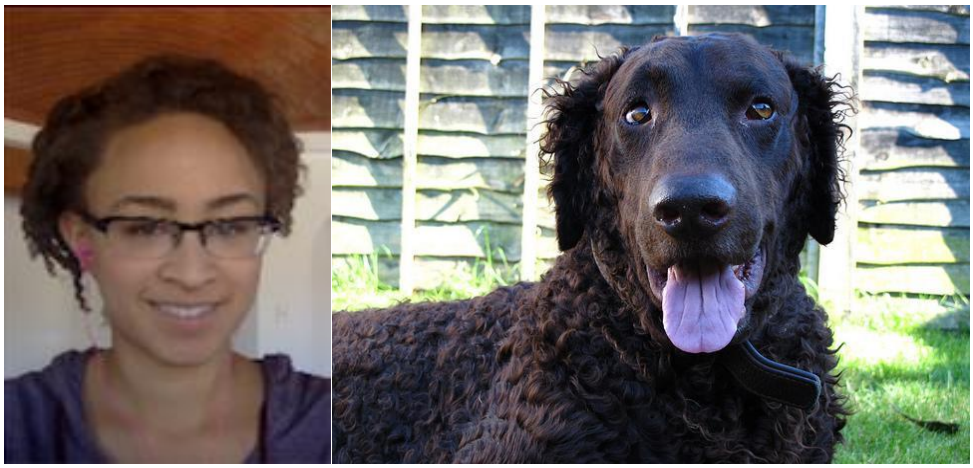
Domain Background:

Its well-known real-world problem. Given an image of a dog we need to identify the breed of dog. If human image is given as an input, we need to find a dog breed resembling dog breed. It's a multiclass classification problem which uses supervised machine learning. I choose this project because it involves work to build an end to end pipeline. I wanted to build a complete application which takes image from UI and show correct prediction.

Datasets and Inputs

The dataset we are going to use here is provided by Udacity. We have 13233 images of human and 8351 dog images. We have 133 classes of dog.

The data is distributed in folder according to the classes so that Dataloader can identify image class based on folder name for the dog data. While for Human data. Images are stored in folders according to Human names.



Sample Images from Datasets

Project Structure

Step1. Import libraries and data.

Step2. Detect Human using OpenCV based Haar Cascade detector.

Step3. Build a VGG16 based dog detector model

Step4. Perform image pre-processing and augmentation and create train, test and validation set.

Step5. Create a CNN to Classify Dog Breeds (from Scratch)

Step6. Create a CNN to Classify Dog Breeds (using Transfer Learning)

Step7. Write an Algorithm to return the following thing based on below three things:

1. if a **dog** is detected in the image, return the predicted breed.
2. if a **human** is detected in the image, return the resembling dog breed.
3. if **neither** is detected in the image, provide output that indicates an error.

Step1. Import Libraries and Data

All the required libraries we installed in first cell of the notebook,

We can download required data from below links to use the

- Download the dog dataset. Unzip the folder and place it in this project's home directory, at the location /dog_images.
- Download the human dataset. Unzip the folder and place it in the home directory, at location /lfw

Step2. Detect Human using OpenCV based Haar Cascade Detector.

We first identify if image is dog image or Human image.

To detect human, OpenCV based Haar Cascade Detector is fast and efficient solution.

```
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
```

If Human is detected this function returns True else, it Return False

This algorithm was able to detect 98% Human Face from Image. I this detection is result is good.

Step3. Build a VGG16 based dog detector model

For Dog Detection we don't have OpenCV based model but VGG16 model is trained on large corpus of data with the classes mentioned in this file.

https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a

It also has classes related to Dogs in the index range 151-268. If VGG16 based classifier predict index in mention it means that dog is detected.

```
# Preprocessing Steps
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:
    transformed_image=transformed_image.cuda()

# Model Prediction
output = VGG16(transformed_image)
torch.max(output, 1)[1].item()  # Getting index from output
```

I simply pre-process it and downloaded VGG16 and used it for prediction.

I was able to get 100% Dog face Detection using this algorithm.

Step4. Perform image pre-processing and augmentation and create train, test and validation set.

We created Three Datasets:

1. Train Set: For Model Training
2. Valid Set: For Hyperparameter Tuning while Training the Model
3. Test Set: For Testing the Model Performance on Unseen Data

There are few pre-processing steps need to be done on the image datasets which can help us to improve performance of model while training and Testing Step. In terms of Inference Speed, Training Time, Model Accuracy etc. Let me explain the steps I have done in this section

1. Normalize Image: We are normalizing images in the specific range of mean and standard deviation of pixel values in the image using the below transformation of Pytorch, It helps us improve inference speed and model training speed

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

2. Resize Image: The Images in our data might not have same size. Any Machine Learning Model needs to have input image of the same size. Most od the Pretrained Machine Learning Model have input image size of 224*224. We can resize image using below function,

```
transforms.Resize(size=(224, 224))
```

3. For Training set I did few Data Augmentation steps as well which can help to improve results on more real data.

```
transforms.RandomResizedCrop(224) # Randomly select/ Crop Image of 224 size
transforms.RandomHorizontalFlip() # Flip Image Horizontaly
transforms.RandomRotation(15) # Rotate Image by 15 degree
```

After This step, We created a Train Loader and Test Loader Objects. Its is an Easy way to load data while training and Testing the model.

Step5. Create a CNN to Classify Dog Breeds (from Scratch)

The Objective of this step is to Train a Simple CNN based network to achieve atleast 10% Accuracy.

We created a simple Convolution Neural Network based network having Following Architecture.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        ## Define layers of a CNN
        self.conv1=nn.Conv2d(3, 64, 3, padding=1)
        self.conv2=nn.Conv2d(64, 128, 3, padding=1)
        self.conv3=nn.Conv2d(128, 256, 3, padding=1)
        self.pool=nn.MaxPool2d(2, 2)
        self.fc1=nn.Linear(28*28*256, 512)
        self.fc3=nn.Linear(512, 133)
        self.dropout=nn.Dropout(0.15)
        self.batch_norm=nn.BatchNorm1d(512)

    def forward(self, x):
        ## Define forward behavior
        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*256)
        x=self.dropout(F.relu(self.batch_norm(self.fc1(x))))
        x=self.dropout(F.relu(self.fc3(x)))
        return x
```

model contains three convolution layers with output size 64, 128, 256 respectively. Each has kernel of 3 with channel 3 and padding 1(to keep image dimension consitance).

Pooling layer of `2*2` size will reduce dimension of image by 2. It will help us to make model training faster.

`ReLU` activation is added in all the layers. `Dropout` of `0.3` value is added in last two dense layers to reduce overfitting.

We selected CrossEntropyLoss() as a Loss Function. We also used Stochastic Gradient Decent as Optimizer Algorithm, optim.SGD(model_scratch.parameters(), lr=0.05). The Reason behind.

We trained model for 15 Epochs with Learning rate 0.05 and we were getting around 13% Accuracy Score

Step6. Create a CNN to Classify Dog Breeds (using Transfer Learning)

The Objective of this step is to Train a Simple CNN based network to achieve atleast 60% Accuracy

We have used here Transfer Learning. We are taking advantage of model trained on Large Corpus of Image Data for Image Classification Task. We got resnet101 from Pytorch Pretrained Model which contains all the layers except last layer. We are not training weights of the pretrained model. We just added one extra Dense Layer on top of Pretrained layers and we are just training Dense layer.

```
model_transfer = models.resnet101(pretrained=True)
for para in model_transfer.parameters():
    para.requires_grad=False
model_transfer.fc = nn.Linear(2048, 133, bias=True)
```

Note: The Dense Layer has output Dimension 133, same as out number of classes in the both network Since we wanted each node to generate score for each class.

We choose the same optimizer and Loss function same as earlier step.

We trained model for 5 epochs and with Learning Rate 0.05

We got around 78% accuracy on the Test Dataset

Step7. Write an Algorithm

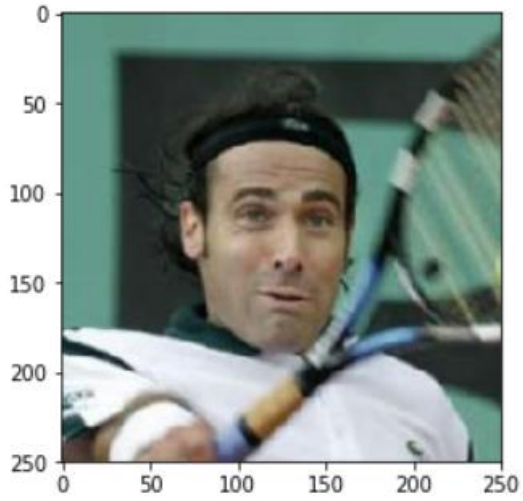
Using simple if else condition if we able to detect face of human we write proper message using a prediction given predict_breed_transfer Function. This function preprocess image and make prediction.

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

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

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

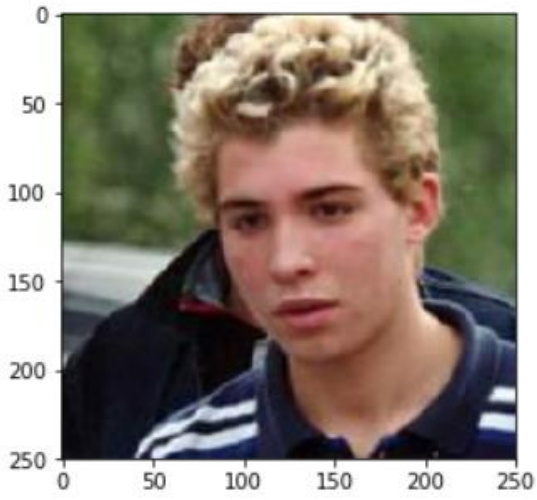
Results:

```
Hello Human!
Predicted breed:  Dachshund
```

```
Hello Human!
Predicted breed:  Nova scotia duck tolling retriever
```




```
Hello Human!
Predicted breed:  Belgian tervuren
```



Justification:

I think the pretrained model we used is giving us better accuracy than expected its giving us 78% accuracy while The Convolution Neural Network we created is giving just 13% accuracy.

This accuracy is considered as good since we are having 133 classes.

Improvement:

The Human detection part can be improved since its also detecting face for dogs with 17% score on 100 images of dog. Maybe we can use some pretrained model like we did with Dog Detection. I trained model using limited data we have; we might need to increase data and do augmentation on it. We can also try different model architecture here.

Reference:

1. *Original GitHub Repository: https://github.com/udacity/deep-learning-v2pytorch/blob/master/project-dog-classification/ Dataset to be downloaded from the link provided in the notebook*
2. *https://medium.com/@fzammito/whats-considered-a-good-log-loss-in-machine-learning-a529d400632d*
3. *Resnet101: https://pytorch.org/docs/stable/_modules/torchvision/models/resnet.html#resnet101*
4. *https://forums.fast.ai/t/image-normalization-in-pytorch/7534/11*
5. *https://pytorch.org/tutorials/beginner/saving_loading_models.html?highlight=eval*
6. *https://github.com/eriklindernoren/PyTorch-YOLOv3/issues/162*