UDACITY

Machine Learning Engineer Nanodegree

Capstone Project

Dog Breed Classifier (CNN)

Vishwajeet Ghatage

June 4th, 2020

Project Overview:

The Dog Breed Identification is popular project in ML/AI community. “Kaggle” also held a competition for this problem. The goal was to build a model capable of doing breed classification of a dog by just “looking” into its image. This problem can be solved by using frameworks like TensorFlow, PyTorch or using a high level approach with Keras.

Here, the capstone project I selected for Machine Learning Engineer Nanodegree is of the same type, and I will be solving it by building two models of Convolutional Neural Network (CNN) in PyTorch, one from scratch and one by Transfer Learning approach using pre trained model.

Problem Statement:

The goal of this project is to create a Convolutional Neural Network, which can be accessed by an endpoint or API to process real-time user provided images and complete following two tasks,

1. **Dog breed identification**: If a dog is present in the image, identify its breed.
2. **Human identification**: Identify human if present and predict the resembling dog breed.

Metrics:

As our training data is imbalanced, it will be better to go with “Categorical Cross Entropy” or “Multi class Log Loss”. As it will comparing actual and predicted labels for loss calculation, it’ll be a good metric.

$$F = -1/N \sum_i^n \sum_j^m y_{ij} * \ln(p_{ij})$$

I have used accuracy metric to evaluate the model on test data.

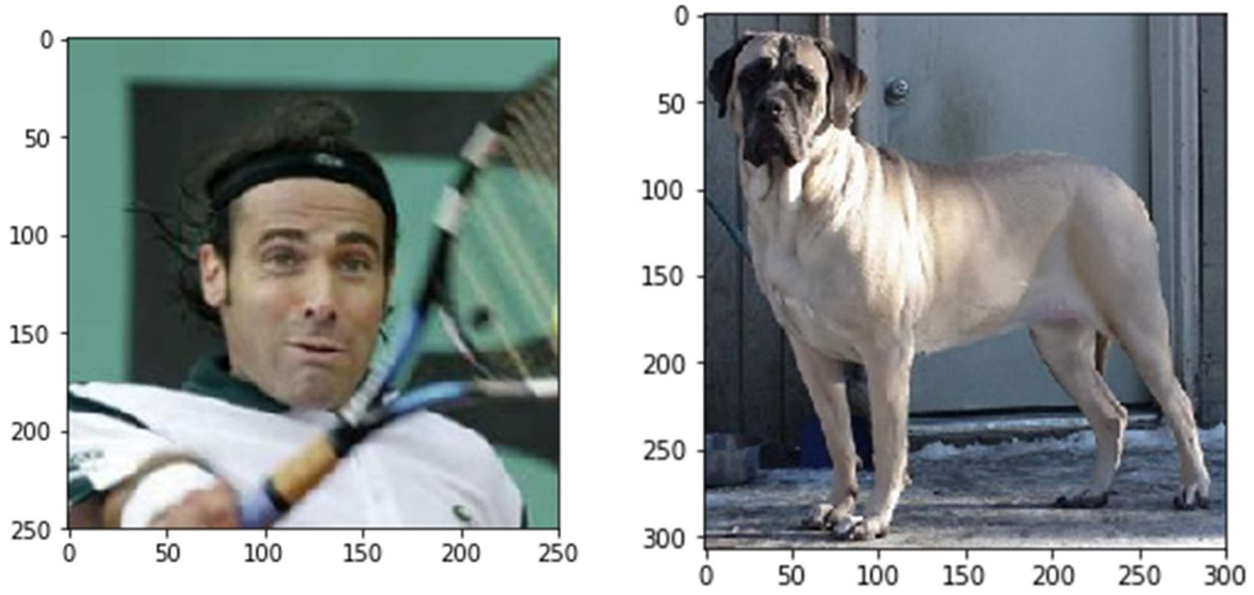
Accuracy = correct classifications / dataset size

Data Exploration:

Input data for training will be an image either containing dog or human. The data is provided by Udacity containing both dog and human images.

Images of Dogs are arranged in three sub directories each for training, validation and testing. Where there are total 8,351 images, 6,680 for training, 835 for validation and 836 for testing. There are total 133 dog breeds provided in dataset, with images having different dimensions. The data is imbalanced as the distribution for each breed is not same.

Similarly, there are 13,233 images of 5,750 people. These images are of size 250 x 250, and these are also not balanced as there are more images of some people comparing to others.



Images form training dataset

Algorithm and Technique:

I will use Convolutional Neural Network (CNN) for the task to identify dog breeds. CNN is a Deep Learning model i.e. type of Artificial Neural Network, which focuses on using convolution operation as it’s more efficient than plain ANN. It requires less parameters (Weights, Biases), and it performs well on the image data.

I have used OpenCV’s implementation of Haar features based cascaded classifier to detect human, a VGG-16 pretrained model for dog detection. I designed a simple CNN model from scratch for dog breed identification, but I used ResNet50 pretrained model with transfer learning to accomplish the task of breed identification.

Algorithm:

```
{
    if dog_is_dected then:
        predict_breed(); // if dog is in the image, predict its breed.
    else if human_is_detected then:
        predict_breed(); // if human is in the image, predict resembling breed
    else:
        report_error; // if both are not in the image, report error
}
```

Benchmark Model:

1. We will build a CNN with trial and error approach, tuning its hyper parameters step by step. The model needs to have accuracy in range at least 10-20% after some epochs of training.
2. The models, created by transfer learning needs to have 70% or more accuracy.

Data Preprocessing:

1. Input **tensor:**
I have set the size of input tensor to 224x224, as it's the standard input size for most of the popular pretrained CNNs.

2. Training data:
To resize training data, I have used RandomResizedCrop(224), and for augmentation I have used RandomHorizontalFlip() and RandomRotation(15) to flip some random selected images horizontally and rotate them in range of 15 degrees.

3. Validation and Test data:
I have used Resize(size=(224,224)) to resize the images and I have not applied any augmentation on validation and test data as we'll not use it in training.

4. Normalization:

I have used Normalize(mean=[0.485, 0.456, 0.406],std=[0.229, 0.224, 0.225]) from PyTorch documentation to normalize the data.

Implementation:

Step 1: I have defined a function to detect human face in image using OpenCV's cascaded classifier.

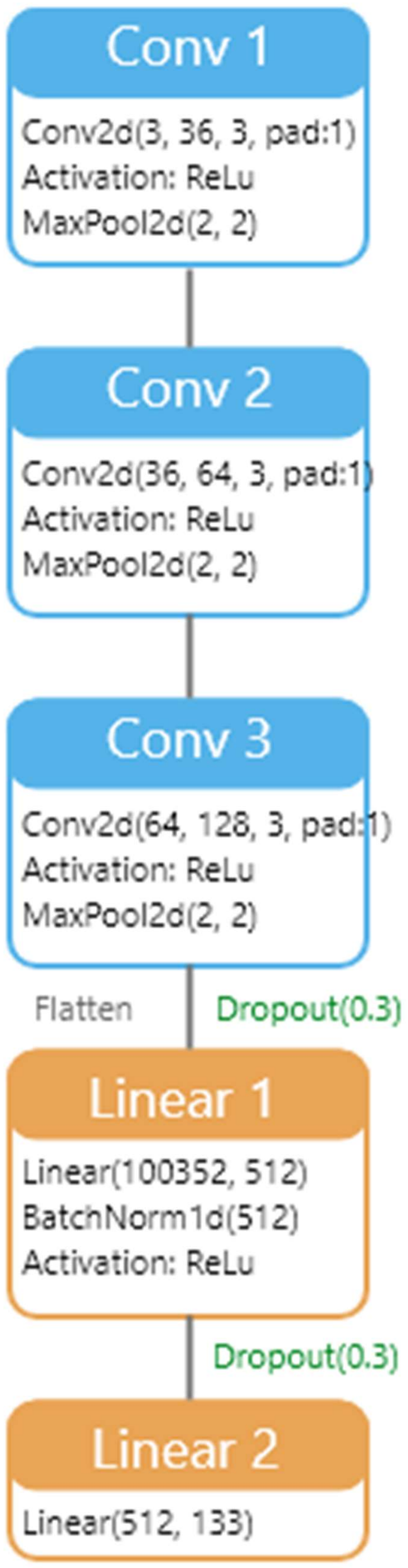
```
# returns "True" if face is detected in image stored at img_path
def face_detector(img_path):
    img = cv2.imread(img_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray)
    return len(faces) > 0
```

Step 2: I have downloaded and used VGG-16 pretrained model to detect dog in image.

```
### returns "True" if a dog is detected in the image stored at img_path
def dog_detector(img_path):
    ## TODO: Complete the function.
    index = VGG16_predict(img_path)
    return 151 <= index <= 268 # true/false
```

Step 3: I have created a CNN model from scratch, to identify dog breed

Step 4: Creating a CNN from scratch, training and evaluating it to identify dog breeds. I have used CrossEntropyLoss() as cost function and SGD(lr = 0.03) as optimizer. The model architecture is,



CNN Model Architecture

Step 5: I have created a ResNet50 model, with transfer learning approach, to identify the dog breed, with CrossEntropyLoss() and SGD(lr = 0.01) optimizer. I have replaced the final fully connect layer of ResNet50 to match our output i.e. total dog breeds (133) and defined a function to get the breed as,

```
def predict_breed_transfer(img_path):
    # load the image and return the predicted breed
    img = Image.open(img_path).convert('RGB')
    transformations = transforms.Compose([transforms.Resize(size=(224, 224)),
                                          transforms.ToTensor(),
                                          normalization])
    img = transformations(img)[:3,:,:].unsqueeze(0)
    if use_cuda:
        img = img.cuda()
    output = model_transfer(img)
    index = torch.max(output,1)[1].item()
    return class_names[index]
```

Step 6: Finally combining these models, I defined a function to accomplish following tasks,

1. If dog is in the image, return its breed.
2. If human is in the image, return resembling dog breed.
3. If image doesn't contain dog/human the output will indicate error.

```
def show_img(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 dog_detector(img_path):
        breed = predict_breed_transfer(img_path)
        print('Dog detected!')
        show_img(img_path)
        print('Predicted Breed:', breed)

    elif face_detector(img_path):
        breed = predict_breed_transfer(img_path)
        print('Hello, human!')
        show_img(img_path)
        print('You look like a', breed)

    else:
        print('ERROR: Invalid Image (Image should contain dog/human)')
        show_img(img_path)
    print('-'*60)
```

Refinement:

The model I have created from scratch meets the benchmark accuracy with 14% accuracy on test data with 20 epochs of training. To improve the performance more, I have used ResNet50 pretrined model, and it reached to 84% of accuracy in 20 epochs.

Model Evaluation and Validation:

1. **Human Detector**: I have tested OpenCV's cascaded classifier on 100 images each of human and dog, where it detected 98% humans correctly and misclassified 17% of dog images as humans.
2. **Dog Detector**: I tested VGG-16 pretrained model on similar data and it detected all dogs correctly and misclassified 1% of humans as dogs.
3. **Dog Breed Detector**: ResNet50 model with transfer learning detected breeds of 703 dogs out of 836 correctly with 84% accuracy.

Justification:

The model is performing better than the benchmarks. As my CNN model has 14% accuracy and the ResNet50 has 84%, but there is more room for improvements.

Reflection:

Completing this project was the way to explore some things which I never had worked on like I got to learn OpenCV usage to detect human faces. Summarizing the project work, I trained multiple CNN models with changing the architecture each to get test accuracy more than 10%.

I read some research articles on this topic and I came to know some helpful tricks to faster the computations while performing training and validation by doing small changes to my code.

Improvements:

The model performance can be improved by using deeper CNN as ResNet101 if the model is under fitting. In our case I think we have less data to train on, we can add more breeds and more images to those breeds. Also we can try with multiple different augmentations. Hyper parameter tuning can be used to set the values of batch size, learning rate, dropout rate etc. and we can also use Ensemble Learning also with multiple parallel models.

References:

1. **Udacity's GitHub repository:**
   https://github.com/udacity/deep-learning-v2-pytorch/tree/master/project-dog-classification

2. **PyTorch documentation:**
   https://pytorch.org/docs/master/

3. **Multi class log loss:**
   https://stats.stackexchange.com/questions/113301/multi-class-logarithmic-loss-function-per-class

4. **Article on solution to similar problem:**
   https://towardsdatascience.com/dog-breed-classification-hands-on-approach-b5e4f88c333e