Udacity Machine Learning Capstone Project

Dog Breed Project Report

Wednesday, October 7, 2020

Abdelrahman Abdelnasser Mohamed

Table of Contents

Definition

Overview

Dog Breeds identification importance can exceed the problem itself. First of all, its a direct examination that can tell how far the accuracy of CNN can reach. It’s a direct application of using CNN in real life application. Furthermore, Identifying the breeds of dogs is important in many other applications. Many features of dogs can vary from a breed to another. Identifying the breed itself is the first step in figuring out many deeper information about the dog. For example, its age, medical state, from the bio metric features could indicate its instantaneous psychological state if he sad, happy, angry ... etc. As stated previously any other further application on extracting whatever information of dogs out of its images is basically dependent on breeds classification. Moreover, Such an application could be further extended to be deployed in a real time application that could help vets to identify dog breeds.

Problem Statement

In this project, I am implementing a CNN that classifies input image of a dogs into their associated breeds. My model classifies 133 different breeds. Two main approaches are used. As a proof of concept, a CNN is built from scratch and trained to achieve 10 % accuracy. This 10 % is larger than accuracy achieved by random guessing. Moreover, to implement the CNN in a real world application, I use transfer learning to transfer the knowledge of the powerful ResNet50 to work in my domain, predicting dog breeds. Finally, I use my model in a function run_app() to work for real world dog images.

Metrics

The metric used to evaluate the model is the accuracy of the classifier. That’s, for the images in the test set, how many are correctly classified and how many are misclassified.

$$\text{Accuracy} = \frac{Number\ of\ correctly\ classified\ images}{Total\ number\ of\ images}$$

Analysis

Data set

The Data set used contains 8351 images for dogs of 133 different breeds. Data is divided into:

training:

6680 images. Nearly 50 images for each class.

validation:

835 images. Nearly 6 images for each class.

testing:

836 images

Moreover, there are 13233 images of Humans. These images are used to examine the detect_faces() function developed at the very beginning of the notebook.

Techniques

As mentioned above. Two approaches are taken. As a proof of concept, a CNN is built from scratch. And to deploy the model, I use transfer learning with ResNet50. The architecture of each approach is described in details in the Methodology section.

Benchmark

The benchmark is determined by Udacity itself. As mentioned in the notebook, the model_scratch is required to obtain at least 10% accuracy. While the model_transfer should obtain a higher accuracy, above 60%

Methodology

Data Preprocessing

Before defining the data loaders, set of transformers are defined to preprocess data. I defined transformers for Train and Test/Valid separately.

Train Transforms:

- preprocessing:

  - Transform data to Tensors

  - Normalize Data

- Data Augmentation:

  - Random Resized Crop

  - Random Horizontal Flip

Valid/Test Transform:

- Preprocessing:

  - Transform Data to Tensors

  - Normalize Data

- Data Augmentation:

  - Center Crop

In addition to that, All images fed to the network have the shape 224 x 224.

Afterwards, train, valid, and test loaders are defined. Each of batch size of 64 and with shuffling.

Note: Same Data loaders are used with the two approaches, The CNN from scratch and the transferred ResNet50.

Implementation

Using Jupyter Notebooks, All the codes and created functions are implemented. All the codes are written in python. Moreover, I used some frame works and libraries during development for many purposes.

- Pytorch: Building, Training, and Validating the CNN
- glob: Read Image Data
- matplotlib: Showing Images

- Open-CV: Reading images
- Numpy: Manipulating nd-arrays
- PIL: Loading Images for testing

During Developments, three helper functions are implemented:

1. face_detector(img_path)

   uses haarcascades to detect humans in an input image.
   Input: image path
   output: True, If contains human. False, otherwise.

2. dog_detector(img_path)

   uses VGG16 to detect dogs in an input image
   Input: images path
   output: True, if contains a dog. False, otherwise.

3. predict_breed_transfer(img_path)

   uses the developed model, using transfer learning, to predict the dog breed
   Input: image path
   Output: Name of the predicted breed

4. run_app(img_path)

   Wrap up the algorithm. Algorithm steps is as follows:

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.

CNN from Scratch Approach

As a proof of concept, A CNN is built from scratch. This CNN was not trained to obtain a high accuracy but to get more than 10%. In this section, I describe the Data Loaders, Architecture the Training procedure, and the results obtained.

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, 16, 3, padding = 1)
        self.conv2 = nn.Conv2d(16, 32, 3, padding = 1)
        self.conv3 = nn.Conv2d(32, 64, 3, padding = 1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(64 * 28 * 28, 5000)
        self.fc2 = nn.Linear(5000, 1000)
        self.fc3 = nn.Linear(1000, 133)
        self.dropout = nn.Dropout(0.25)


    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)))

        #flatten image
        x = x.reshape(-1, 64*28*28)
        x = self.dropout(x)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        x = self.dropout(x)
        x = self.fc3(x)

        return x
```

I used the same architectue used while developing the network used on CIFAR in the Extracurricular materials. However, I obtained low test Accuracy. That's why I added one more Fully connected layer. I believe that the added layer helped in adding more relaxation while mapping the features extracted by the convolutional layers to the 133 classes.

Before adding the third Fully connected layer:
the 50176 output features were reduced directly into 500 features only!

Input --> Convlayer --- 50176 different output
50176 --> FC1 ---> 500
500 --> FC2 -→133

After adding the third Fully connected layer:

Input --> Convlayer --- 50176 different output
50176 --> FC1 ---> 5000
5000 --> FC2 -->1000
1000 --> FC3 -→133

Moreover, I used Max pooling, which is powerful against small translations and helpful in reducing sizes. In addition to applying dropout to reduce over fitting

Training

I used Cross Entropy loss as the criterion . Furthermore, a learning rate = 0.07 Stochastic Gradient Descent is used to optimize the Network parameters. Finally model is trained for 25 epochs

Results

Model is trained to obtain:

```
Training Loss: 3.472678

Validation Loss: 3.729341

Test Loss: 3.743663

Test Accuracy: 13% (114 dog images classified correctly out of 836 images)
```

Transfer Learning Approach

I used the powerful ResNet50 that was trained on the large ImageNet dataset. ResNet50 contains punch of well trained convolutional layers that's capable of extracting deep features of images. Moreover, I replace the final fc layer that map extracted features to label 1000 different classes to only contain 133 node as same as my dog breed classes.

Architecture

I used the architecture of the ResNet50 as it is. I only added the last Fully connected layers to map the extracted 2048 features produced by the convolutional layers to 133 dog breed classes.

Training

Cross Entropy Loss is used as a criterion. Moreover, Adam is used with learning rate = 0.003 to optimize only the parameters of the modified final Fully connected layer. Finally, the model is trained for 20 epochs.

Results

Training:

Best validation loss was obtained after 4 epochs:
Training Loss: 1.140765
Validation Loss: 0.607286

Testing:

Test Loss: 0.768759
Test Accuracy: 80% (675 images are correctly classified out of 836 images)

Results

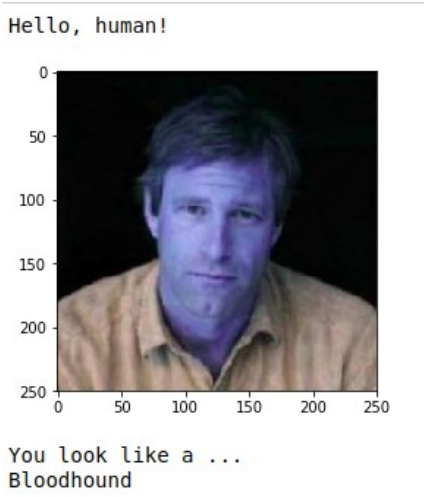
All the developed work is wrapped up in `run_app()` function that takes the path to an image as an input ant runs the algorithm on it. The Algorithm implements is as follows:

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.

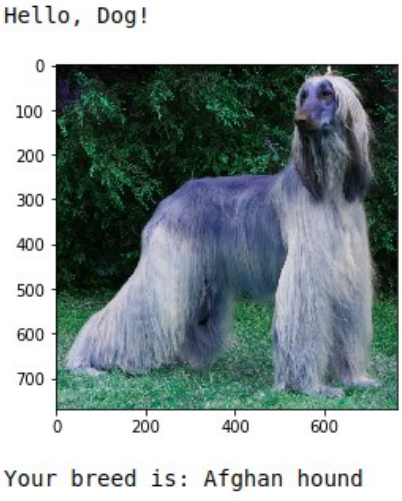
I use early developed `face_detector()` and `dog_detector()` functions. In addition to that, I use the transfer trained model, the one with accuracy 80% , to detect the dog breed of the input image. Finally, I tested the algorithm on different functions and the results are as follows:

For Human input image:

s



For Dog Input image:



References

[1] https://mc.ai/dog-breed-classification-using-cnn/

[2] https://towardsdatascience.com/deep-learning-build-a-dog-detector-and-breed-classifier-using-cnn-f6ea2e5d954a

[3] https://towardsdatascience.com/dog-breed-classification-using-cnns-f042fbe0f333

[4] https://gist.github.com/jkarimi91/d393688c4d4cdb9251e3f939f138876e