Capstone Project Report: Machine Learning Engineer Nanodegree

Dog Breed Classifier using CNN

Srijan Sahay
October 24th, 2020

Domain Background

Image classification is one of the fundamental and the trending topic in the field of Machine Learning and Deep Learning. This also include real world image processing and computer vision problems. Today, researchers are trying to classify different objects in an image. Classifying of animal and its breed is also one of them and is highly appreciated among animal lovers. In this project, we will be using Convolutional Neural Network (CNN) to predict the breed of the dog in the given image. Also, if a human image is given, we will be predicting the label that closely resembles dog breed.

Problem Statement

The primary goal of this project is to do the following tasks using CNN:

i. Given an image of the dog, the model should predict its breed
ii. If human is detected in image, the model should predict resembling dog breed

Datasets and Inputs

For this project, the dataset is provided by Udacity. The dataset contains the image of dogs and humans.

Dog Dataset: The dog dataset has 8351 images and is saved in the location /dog_images. The dimension of each image is not same. We have 133 types of dog breed and the dataset is imbalanced

Human Dataset: The human dataset has 13233 images and is saved in the location /lfw. The dimension of each image is same i.e. 250X250

Data Exploration

There are 13233 total human images, which can help us train a model to identify the human image. And there are 8351 total dog images for dog classification.

For human face images, they all have the same dimension: 250×250. Images have different background and different angles. The data is not balanced because we have 1 image for some people and many images for some.

For dog images, the images are of different sizes and different backgrounds, some images are not full-sized. The data is not balanced because the number of images provided for each breed varies. Few have 4 images while some have 8 images.

Data Pre-processing

For creating any machine learning or deep learning-based model, the foremost step is to pre-process the dataset to improve the results of training and create a robust feature sets than can accurately solve a given problem. Following are the steps that are involved in data pre-processing:

1. Resize the images in the dataset to a fixed size of 224*224 pixels.
2. Perform augmentation like flips and rotations of the images to include more variation in the dataset.
3. Normalize the pixel values of the images.
4. Convert the images into appropriate tensor the feed it in the model for training.

Implementation

We have divided this project in several steps like first performing human face detection and dog detection to extract human and dog region of interest (ROI).

Detect human image using OpenCV which in turn uses a Haar feature based cascade classifiers.

Create a dog detector using a pretrained VGG-16 model. VGG-16 is one the CNN architecture which was trained initially on ImageNet dataset.

To test the performance of the VGG-16 for the dog detection, we tested the first 100 human images and the first 100 dog images and we found that 0.02% of the human images have dog and 100% of dog images where detected for dog.

We trained a custom model to predict breed of the dog given an input image. For this we use CNN model.

Finally, we train a CNN model using transfer and compare the results.

To summarize, the architecture of custom CNN model is as follows:

```
Net(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv5): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (dropout): Dropout(p=0.2)
  (conv_bn1): BatchNorm2d(224, eps=3, momentum=0.1, affine=True, track_running_stats=True)
  (conv_bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv_bn3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv_bn4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv_bn5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv_bn6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc1): Linear(in_features=12544, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=133, bias=True)
)
```

Further, dropout hyperparameter is used to avoid overfitting. However, this custom CNN model performs poorly, and accuracy obtained on test dataset is just 23%. To improve the performance transfer learning technique is used.

For transfer learning, Resnet50 architecture is used and initial layers are fined tuned. Then, this model was trained for 15 epochs and there is significant improvement in the result. The accuracy

obtained is 81%. This really shows that if we have small dataset it is better to used transfer learning technique than training the custom CNN model.

Benchmark

As we have comparatively less dataset for this project, the custom CNN model will have less accuracy. The CNN model created from scratch must have accuracy of at least 10%. This can confirm that the model is working because a random guess will provide a correct answer roughly 1 in 133 times, which corresponds to an accuracy of less than 1%. Moreover, we will be also using transfer learning technique to improve the performance of this model. Thus, the benchmark for creating CNN model using transfer learning should be at least 60% and this much higher than a custom CNN model.

Formula for **Accuracy = Number of items correctly classified/ All classified items**

Model Evaluation

As we have used 3 models in total, the result for them are as follow:

1. Human Face Detector: 98% accuracy in human dataset and 17% accuracy in dog dataset
2. Custom Dog Classifier: 100% accuracy for dog dataset and 1% for human dataset
3. Resnet50 model: Custom dog breed classifier model gave 81% accuracy

Justification

I think the model performance is better than expected and the performance is appreciable for a complex real-world problem. The model created using transfer learning have an accuracy of 81% compared to the CNN model created from scratch which had only 23% accuracy.

References

Original repo for Project - GitHub: https://github.com/udacity/deep-learningv2pytorch/blob/master/project-dog-classification

Pytorch Documentation: https://pytorch.org/docs/master/

Resnet50: https://pytorch.org/docs/stable/_modules/torchvision/models/resnet.html#resnet50

ImageNet training in Pytorch: https://github.com/pytorch/examples/blob/97304e232807082c2e7b54c597615dc0ad8f6173/imagenet/main.py#L197-L198