Machine Learning Engineer Nanodegree

Capstone Project

I. Definition

Project Overview

This project is based on AI neural network, which shows very promising accomplishment recent years and launch a revolution on technology.

Computer vision is one of fields in AI. A very important net in Computer Vision is convolutional neural network. In fact, in ImageNet competition, models that using convolutional neural network surpassing human-level performance, that is a great accomplishment in the human histories!

So, convolutional neural network is a very important net, and Computer Vision is a very promising domain.

In this project, I will learn how to build a pipeline to process real-world, user-supplied images. Given an image of a dog, my algorithm will identify an estimate

of the canine’ s breed. If supplied an image of a human, the code will identify the resembling dog breed.

dog dataset come from https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip. There are 8351 total dog images, 133 classes.

human dataset come from https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip. There are 13233 total human images, 5749 total different human.

Problem Statement

Given an image of a dog, identify an estimate of the canine’ s breed. If supplied an image of a human, the code will identify the resembling dog breed. when identify dog breed, solution must attain at least 60% accuracy on the test set.

I will use OpenCV's implementation to detect whether human faces in images, model pre-train on ImageNet to Detect Dogs and to train a CNN to Classify Dog Breeds.

Metrics

The main probleam here is to correctly identify dog breed, this is a classification problem , so I will use accuracy as our evaluation metrics, which is correct_num / total_num

correct_num: the number of correct classification in test dataset

total_num: the number of total images in test dataset

II. Analysis

Data Exploration

Dog dataset

This dataset should contain 8351 total dog images. And 133 folders, each corresponding to a different dog breed, so there are 133 classes, each image’ s size is different, but all images have 3 channel which is RGB.

Suppose num_per_class_list represent numbers of images in each class, then this list has mean 62.79, standard deviation 14.80, min 33, max 96. so, this is a little imbalanced dataset.

This dataset is input to the model to train to classify dog breed.

A sample of image is below:

Human dataset

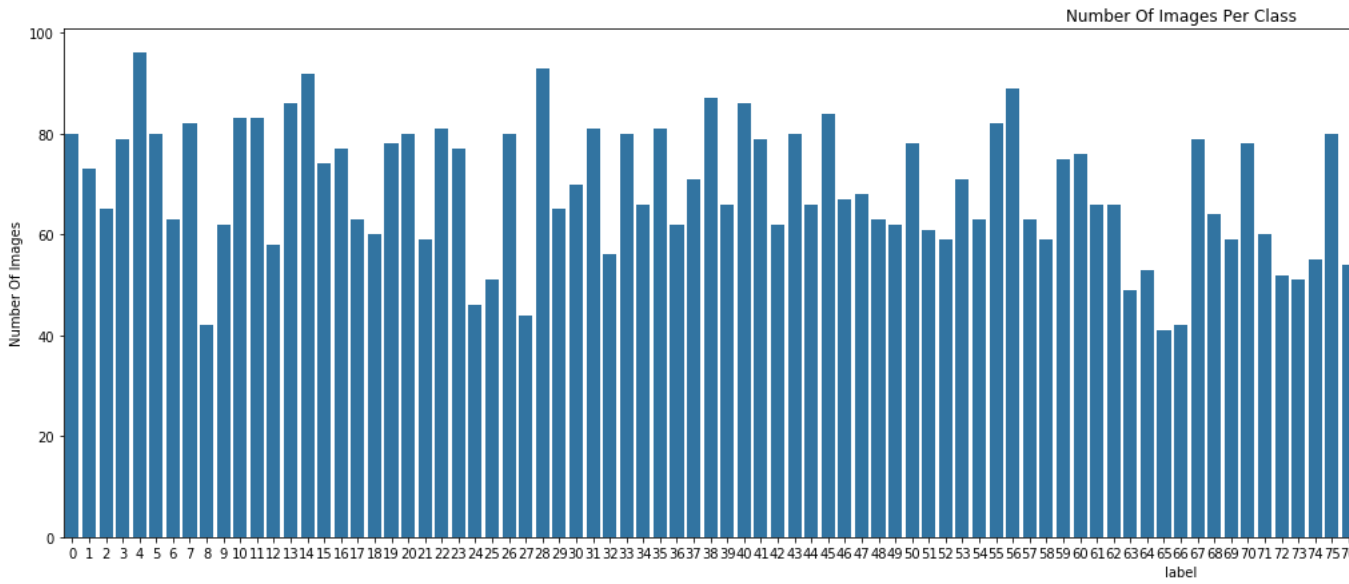
There are 13233 total human images, 5749 total different human. Each image' s size is different, but all images have 3 channel which is RGB.

We use OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images, so this dataset is just for test.

A sample of image is below:

Exploratory Visualization

The main problem is to classfy dog breed, so I visualize number of images per class from dog dataset.



From the visualization, we can see that dataset is a little imbalanced, but, just a little. Most class have at least 50 images, the min is around 40. So, good enough dataset.

Algorithms and Techniques

detect human faces

OpenCV provides many pre-trained face detectors, stored as XML files. We use OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images.

About Haar feature-based cascade classifier’ s Theory, can be found at https://docs.opencv.org/trunk/db/d28/tutorial_cascade_classifier.html

Classify Dog Breed

We use a CNN to classify dog breeds (using Transfer Learning), specifically, we will use VGG16(pre-trained on ImageNet) as original model to train to classify dog breeds.

We only have 8351 dog images, which I think is a small datasets. but VGG16 pre-trained on a very large dataset which have many dog images, also VGG16 achieve excellent performance on ImageNet. So, if we use pre-trained model to do transfer learning, the final result should be good. Hence, transfer learning using model pre-trained on ImageNet is worth a shot. We just need to replace the final linear layer with out_features=133(number of classes in dog dataset), and train the network to update the weights of the new fully connected layer would be OK for this situation.

Benchmark

The task of assigning breed to dogs from images is considered exceptionally challenging, so we use a random-guess-model as a benchmark model. Dog dataset have 133 classes, so random-guess-model have an accuracy of 1/133≈0.75%

III. Methodology

Data Preprocessing

When using pre-trained model, input images need to be normalized, i.e. mini-batches of 3-channel RGB images of shape (3 x H x W), where H and W are expected to be at least 224. The images have to be loaded in to a range of [0, 1] and then normalized using mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225].

Implementation

Detect Humans

detech a face in image can be split into following steps:

- extract pre-trained face detector
- read in a image
- convert color image to grayscale
- use pre-trained face detector to find faces in image

Detect Dogs

VGG16 model can accepts an image as input and returns the index corresponding to the ImageNet class, the output should always be an integer between 0 and 999, inclusive. And categories corresponding to dogs appear in an uninterrupted

sequence and correspond to dictionary keys 151-268, inclusive. Thus, in order to check to see if an image is predicted to contain a dog by the pre-trained VGG-16 model, we need only check if the pre-trained model predicts an index between 151 and 268 (inclusive).

Also, All pre-trained models expect input images normalized to be mini-batches of 3-channel RGB images of shape (3 x H x W), where H and W are expected to be at least 224. The images have to be loaded in to a range of [0, 1] and then normalized using mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225].

So, Detect Dogs can be split into following steps:

- get VGG16 model
- read in a image
- pre-process image to tensors as described above
- put processed tensor into model to get output
- if the pre-trained model predicts an index between 151 and 268 (inclusive) then a dog is detected in the image

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

We use VGG16, and as mentioned above, image should be processed before put into VGG16. For validation set, test set, image will be pre-processed as mentioned above. But when pre-process training set, I also augment the trainning data by random rotation,random resized crop,random horizontal flip besides previous mentioned process.

Anyway, to train a model to classify dog breeds can be split into following steps:

- loading, process and transform images to appropriate data structure(DataLoader in this situation)
- get VGG16 model
- replace the final linear layer with out_features=133(number of classes in dog dataset)
- freeze all the params in the model except for final linear layer
- move model to GPU if GPU is available
- specify a loss function and optimizer
- train the model and save the best model during training
- test the model using test dataset

Refinement

Because test accuracy on classify dog breeds using transfer learning is 82%(exceed 60%) which is good enough, so, I haven’ t do the refinement.

IV. Results

Model Evaluation and Validation

The final model has test loss: 3.207401 and test accuracy: 82% (688/836), it meets the expectation in terms of test accuracy. Because during training, model doesn’ t see the test data, so this means model generalize well enough.

I also re-run the whole training process, and get test accuracy: 84% (704/836), because we randomly augment our training data and shuffle traning data at every epoch, it means small perturbations in training data or the input space can’ t greatly affect the results. Thus, our model is robust.

Justification

Benchmark has accuracy around 1%, and the final model achieve 82%, so final model is way better than the benchmark.

Because we use pre-trained model VGG16 and transfer learning technique, we only replace final layer in the VGG16. It means the model should perform excellent performance before final layer. Then we only train the final linear layer, it means

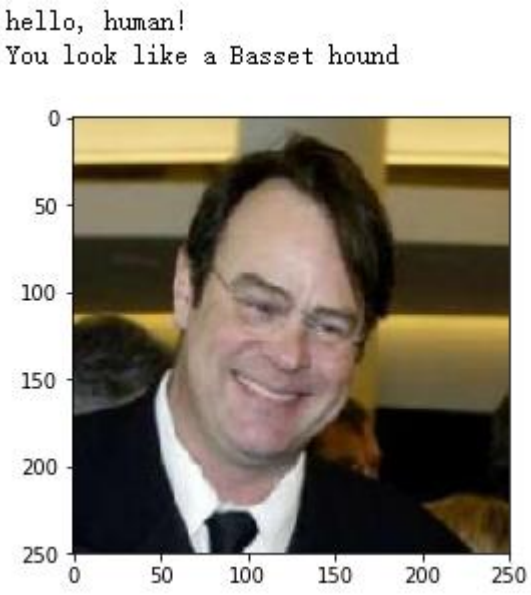
we can get the excellent performance from original VGG16. Thus, our final model should be robust.

Considering that even a human would have trouble distinguishing different dog breeds in many situation, so, I think 82% accuracy is well enough to solved the problem.

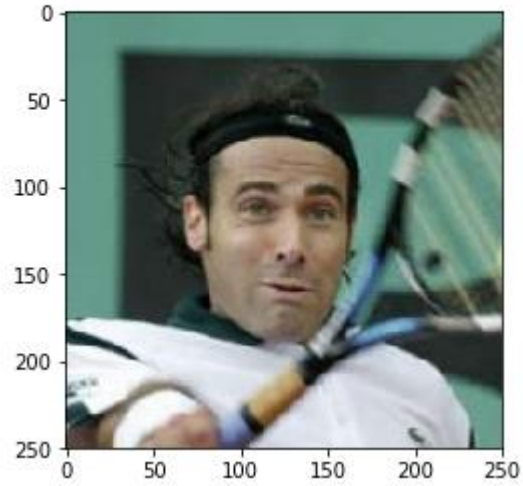
V. Conclusion

Free-Form Visualization

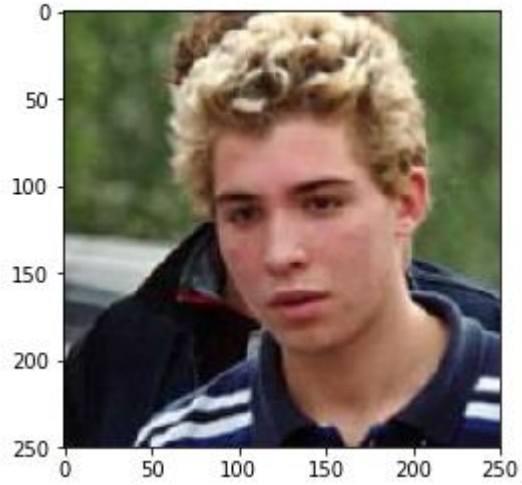
I have test the result algorithm on sample images:



hello, human!
You look like a Australian shepherd

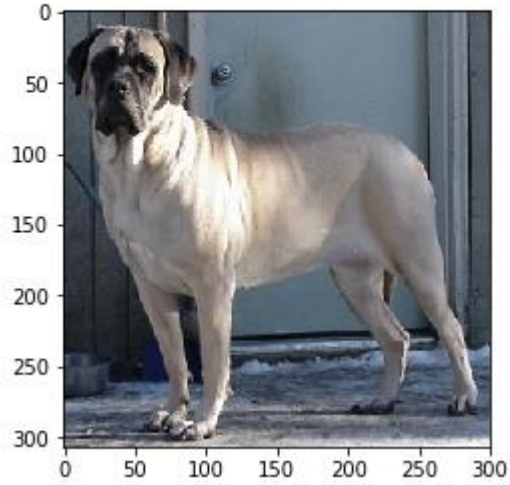


hello, human!
You look like a Portuguese water dog

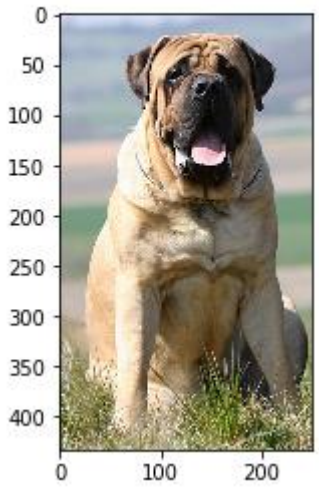


hello
look like a Mastiff

hello
look like a Mastiff



```
hello
look like a Bullmastiff
```



we can see that result algorithm can detect human and dog, and produce the corresponding dog breed. very well!

Reflection

In this project, I

- import datasets
- use OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images (write face_detector() function)
- assess the human face detector, and get a pretty good result
- use pre-trained VGG-16 model to detect dog in an image
- assess the dog detector, and get a pretty good result

- Create a CNN to Classify Dog Breeds (from Scratch)
- Create a CNN to Classify Dog Breeds (using Transfer Learning)
- write algorithm to identify an estimate of the canine’ s breed (If supplied an image of a human, the code will identify the resembling dog breed.)
- test algorithm on some sample image

There is one difficulty I encounter when try to create a CNN from scratch (not using transfer learning), I only get 1% test accuracy. The validation\training loss seems stop to decrease after train a couple of epochs. And I try smaller CNN or larger CNN, neither get better test accuracy, and I just don’ t know what’ s going wrong at that time. The other day, I realize it may be the learning rate too high, otherwise the traning loss should decrease, so I try a smaller learning rate, and finnally, the test accuracy achieve 14%, much better than 1%.

The final model(obtained using transfer learning) get 82% test accuracy, which I think is very good result. So, if there were a web app, I think this model can be used.

Improvement

I think this final result could be improved:

- I could use transfer learning to implement face_detector(), to try to get a better performance.
- use lower learning rate to train transfer learning model to get a better performance.
- train more epoch
- try many more different pre-trained model to classify dog breeds, and use the best one as final result
- use more technique to augment dataset, so that final model could generalize more well

---

VI. Reference

1. https://pytorch.org/docs/stable/torchvision/models.html
2. https://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html
3. https://github.com/udacity/deep-learning-v2-pytorch/tree/master/project-dog-classification
4. https://github.com/udacity/deep-learning-v2-pytorch/blob/master/project-dog-classification/README.md
5. https://github.com/udacity/deep-learning-v2-pytorch/blob/master/project-dog-classification/dog_app.ipynb