Capstone Project

Phyo Min Thant

Machine Learning Nanodegree

October 3, 2020

Dog Breed Classifier

Definition

Project Overview

The application of computer vision is very broad can be utilized in many fields like public security, healthcare, automotive, agriculture, industrial etc. And with the help of deep learning, much better performance and cloud computing like SageMaker and Azure, computer vision become much more practical and can even be used in real-time. As I prefer computer vision than other machine learning fields, I decided to choose dog breed classifier.

This project will be part of the mobile or web app that can predict dog breed. Moreover, this can also detect human and can say most alike dog breed that can joke around among friends.

Problem Statement

The project will detect dog and human and predict dog's breed. The steps involved are…

1. Import dog dataset and human dataset,
2. Detect humans
3. Detect dogs
4. Create a CNN to Classify Dog Breeds (from Scratch)
5. Create a CNN to Classify Dog Breeds (using Transfer Learning)
6. Write your Algorithm
7. Test Your Algorithm

The final output will be imported image and print out that the input image is human or dog and dog's breed even if the detect image is human.

Metric

Like any other machine learning approach, the dataset is split up into train, validate, and test dataset. The dataset I got is already split up and so I didn't need to do it manually. These spits used in both scratch and transfer learning. The evaluation metric is the accuracy. For getting the best trained model during training, I just compared the best loss with the validation loss and assumed as best trained model so far if validation loss is less than best loss.

Best Loss = min (Best loss, Validation Loss)

Accuracy = corrected prediction / dataset size

Analysis

Data Exploration

Human Dataset

In human dataset, there is 13233 images of 5750 people with 250*250 pixels in all images. Some people got only one image and some got much more images. Because of we only want to detect human and only going to use OpenCV library, I won't split it into train, valid and test.

Dog Breed Dataset

Because the major goal of the project is to classify dog breed, I'll create CNN both from scratch and transfer learning. And that is why, Udacity pre-modified the dataset with train, valid and test splits. In this dataset there is 8351 dog images of 133 breeds that is 6680 images in train, 836 images in valid and 835 images in test. But the image sizes are not identical for each image.

Sample Human images

Sample Dog Images

Algorithms and Techniques

Firstly, I used OpenCV's implementation of Haar feature-based cascade classifiers to detect human in an image. OpenCV provide many haarcascades that is stored in github and I downloaded haarcascade_frontalface_alt.xml and stored it in harcascade folder and detect images with detectMultiScale method. Then, I detect dog with pretrained using pretrained VGG16 architecture that is trained on imagenet and I found that it has over 100 class trained for dogs. After that, I implemented CNN on my own for multiclass classification of dog breed. Then because this model barely classify that is about 12% and so I used Resnet 101 with transfer learning for breed classification.

Benchmark

Udacity set threshold for minimum 10 percentage accuracy in CNN that created by ourselves as classification of dog's breed is very difficult even for us and we will need bigger and better network like imagenet winners. And for transfer learning the minimum threshold is 60 percentage of accuracy.

Methodology

Preprocessing

In implementation of CNN, the first step in preprocessing is resizing to (224 * 224) because I just followed up to the architecture of VGG16 that is gradually decreased the image size into half by half and finally reached to (7 * 7) in the fully connected layer. Also, I need (224 * 224) in transfer learning as resnet101 that I used is trained with this image size. I also add some augmentation techniques to relief overfitting. The augmentation techniques I used are RandomHorizontalFlip and RandomRotation. Finally, all images are converted to tensor to be able to train.

Implementation

I built a CNN architecture for multiclass classification; this architecture is based on VGG16 architecture. I just decreased the image by half in every layer to fully connected layer. I used 64, 128, 256 and 512 filters respectively in layers. I set kernel size 3 and padding 1 for convolutions layers and set kernel 2 and stride 2 in maxpooling to reduce image size in half. And for non-linear activation function, I used RELU. I created 3 fully connected layers and the final layer is to yield predicted class. I also added dropout with p=0.5 to avoid overfitting. I trained the model 30 epochs and the validation loss is not that good even training loss gradually reduced. I wanted to train the model much more may be 100, 200 epochs, but even one epoch take too long and so I got that I need to refine with transfer learning.

Refinement

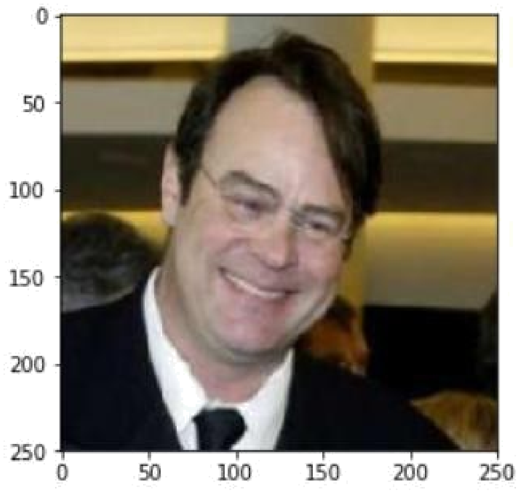
At first, I just picked up VGG16 for transfer learning for the sake of simplicity. But after a lot of trails, I found out Resnet 101 can give the best result for me and I could even get good validation loss on epoch 10. So, I choose this and I got 89 percent accuracy that is much better than threshold. I think this is the pretty good result for this fairly small dataset that is only 8k images for 133 class. I just modify a bit for training the pretrained model for my dataset. I set parameter.require_grad = False and modified the output channel of fully connected layer to 133 that is the number of class in my dataset.

Result

Sample Results

For final result, I'll show the input image to the user and say 'Hey Guy!! You look like a …….' if the model detects human

```
Hey Guy!!
```



```
You look like a ... Welsh springer spaniel
```

If the model detect dog, it'll say 'This dog is ……'

```
This dog is Mastiff breed
```

If it doesn't detect a dog or human, it'll say 'Sorry! We do not detect a human or a dog in this image'

Model Evaluation and Validation

Human Detector: In human detection using OpenCV, I got 104 detection in 100 images (may be two or more detection in some image). But I got 20% of fall short in dog image wrongly detected as human. Nevertheless, it is the good method to detect human.

Dog Detector: From directly prediction from pretrained VGG16 model, I got 100 percentage dog detection and only one percentage of human detected as dog.

CNN-Scratch: In my own CNN model, the model's performance is not great. Although it decreased training loss in mostly every epoch, the validation loss isn't wandering around 4.0 to 4.3 until the training finished. I only got 12 percent accuracy

CNN-Transfer Learning: As the pretrained model that I chose is trained on imagenet, I can learn much better than mine. The training loss and validation loss are not that much different and I could see validation loss decreasing in every epoch. I got 0.37 validation loss in epoch 10 and stopped training. The accuracy is about 89 percentage.

Justification

After finished my project, I realized the power of transfer learning and model from transfer learning can beat easily to my scratch-model even it takes much shorter time.

Conclusion

To conclude my report, with the combination of OpenCV and CNN, we could create a pretty neat object detector and classifier. And with transfer learning, we could save our time and could develop much better classifiers efficiently.

Improvements

I'll to build my project with bounding boxes to the objects by using ssd or yolo rather than just detect and don't know where is the object. The next improvement may be customized CNN architecture instead of just using transfer learning only, I'll add more layer to my own and integrate with transfer learning like using in SSD. I'll add more data to datasets to get better accuracy and avoid overfitting.

References.

1. Write an Algorithm for a Dog Identification App from Kaggle
   https://www.kaggle.com/subhagatoadak/dog-breed-classifier-udacity/notebook

2. Liu W. et al. (2016) SSD: Single Shot MultiBox Detector. In: Leibe B., Matas J., Sebe N., Welling M. (eds) Computer Vision – ECCV 2016. ECCV 2016. Lecture Notes in Computer Science, vol 9905. Springer, Cham

3. Resnet101

   https://pytorch.org/docs/stable/torchvision/models.html?highlight=torchvision%20models

4. OpenCV tutorial

   https://docs.opencv.org/master/d9/df8/tutorial_root.html