Capstone Project

Machine Learning Engineer Nanodegree

Zain Ullah Muhammad
June 17, 2020

Dog Breed Classifier

Definition

Project Overview

In this project we will train a convolutional neural network (CNN) on different dog breed images, and then on passing a picture of dog, it will be able to predict the class of breed. And if human picture is given to our CNN model, it will identify it as human but also it will try to match a similar dog breed that closely matches the picture provided.

The datasets of this project is provided by udacity

Problem Statement

We will approach this task by combining pretrained model and our custom model. The steps we will follow are as follows:

1) Use pre-trained Haar Cascade classifier to identify human faces [Haar cascade OpenCV]

2) Use pre-trained VGG-16 Model to identify dogs in pictures [VGG-NET]

3) Create a CNN model from scratch to identify dog breeds

4) Create another CNN model with resnet50 or similar model using concept of transfer learning to be able to predict more accurately

5) If human picture is provided, identify as human face and predict similar dog breed

6) If dog picture is provided, identify the breed of the dog

7) If no dog or no human is found in picture, then text is returned stating that “The image doesn't contain dog or human”

Metrics

Precision will be used to see how precise our Haar Cascade classifier can identify human faces and not human faces (dog faces).

Precision = True Positive / (True Positive + False Positive)

VGG-16 will be evaluated by comparing the proportion of human detected faces with overall human faces and proportion of dog faces with overall dog faces

Example:

Dog sample images: 100 Images

Dog detected in sample images: 97

Proportion = 97 / 100 = 0.97%

Accuracy will be used as metric to measure efficiency our custom created networks

Accuracy = True Positive + True Negative / Size of all train images

Analysis

Data Exploration

We are provided 2 datasets, one of them contains human images and the other data-set contains dog images. There sizes are as follows (13,233 human images and 8,351 dog images)

```
import numpy as np
from glob import glob

# load filenames for human and dog images
human_files = np.array(glob("/data/lfw/*/*"))
dog_files = np.array(glob("/data/dog_images/*/*/*"))

# print number of images in each dataset
print('There are %d total human images.' % len(human_files))
print('There are %d total dog images.' % len(dog_files))
```

```
There are 13233 total human images.
There are 8351 total dog images.
```

The human faces folder structure is in such a way that every folder has a person name, and every person has an image inside the folder. These folder names are the classes and can be used for predicted index matching, but in our case we won’t be using these classes.

The dog images folder contains 3 main folders (train, test and valid). Each of these folders have sub folders with names of dog breeds, every breed has multiple images on which our model can train. In our case we have 133 sub folders for a certain set “train”, so when we load this data in our code, it will save each folder as python list, and later we will use this list to match our predicted index with the class label.

Images in both folders are colored images and we will keep them this way for our purpose. Ideally we might convert them to gray scale for better results and faster training. I have also resized the images to 224x224 for all dog images so that they will have a uniform size

Human faces folder structure on the left and Dog faces folder structure on the right

Methodology

Data Preprocessing

We will resize first our training images to 256x256, then we will apply random cropping to our image and get only part of image with size 224x224, this size will be the input to our network, we will then convert our image to tensor so that our PyTorch model can understand the values and work on them, we will also apply standard normalization of values (mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

It is a common practice to use this mean values as it has been tested on million of images [StackOverflow Reference]

We will then pass this data to our network in batches so we need to specify batch size, for this case I have tried multiple batch sizes (64,32,16, and 20), In our case I found 20 to be good batch size as the network started learning better. Such a small batch size was also good between our dataset is small

Implementation & refinement

Loss Function

I have chosen the loss function to be CrossEntropyLoss as our tasks involves classification and CrossEntropyLoss is useful in such scenarios, I haven’t tried other loss function on this task.

Optimizer

I tried with Adam optimizer with different learning rates 0.01,0.001,0.05,0.08,0.06 but the network was learning very very slowly, so I switched to SGD optimizer and saw improvement, that is why I decided to stick with SGD for this specific task, I have added weight_decay parameter in SGD to effect for l2 regularization. I tried value 1e-5, but settled with 1e-3

Test Cases

Some test screenshots are provided below:

Optimizer: Adam

Learning rate:0.01

Batch size: 64

Conclusion: Under fitting, network is not learning

```
Epoch 1, Batch 1 loss: 4.897825
Epoch 1, Batch 101 loss: 1186.683472
Epoch: 1 	Training Loss: 1141.662964 	Validation Loss: 4.873906
Epoch 2, Batch 1 loss: 4.867238
Epoch 2, Batch 101 loss: 4.873923
Epoch: 2 	Training Loss: 4.873126 	Validation Loss: 4.871349
Epoch 3, Batch 1 loss: 4.863626
Epoch 3, Batch 101 loss: 4.870667
Epoch: 3 	Training Loss: 4.871250 	Validation Loss: 4.879930
Epoch 4, Batch 1 loss: 4.917325
Epoch 4, Batch 101 loss: 4.868186
Epoch: 4 	Training Loss: 4.868907 	Validation Loss: 4.862080
Epoch 5, Batch 1 loss: 4.850250
Epoch 5, Batch 101 loss: 4.867385
Epoch: 5 	Training Loss: 4.868123 	Validation Loss: 4.867347
Epoch 6, Batch 1 loss: 4.873003
Epoch 6, Batch 101 loss: 4.868076
Epoch: 6 	Training Loss: 4.867275 	Validation Loss: 4.880541
Epoch 7, Batch 1 loss: 4.856688
Epoch 7, Batch 101 loss: 4.867625
Epoch: 7 	Training Loss: 4.867382 	Validation Loss: 4.872171
Epoch 8, Batch 1 loss: 4.849103
Epoch 8, Batch 101 loss: 4.866826
Epoch: 8 	Training Loss: 4.867235 	Validation Loss: 4.847689
Epoch 9, Batch 1 loss: 4.841872
Epoch 9, Batch 101 loss: 4.867584
Epoch: 9 	Training Loss: 4.867157 	Validation Loss: 4.875931
Epoch 10, Batch 1 loss: 4.872229
Epoch 10, Batch 101 loss: 4.866828
Epoch: 10 	Training Loss: 4.866965 	Validation Loss: 4.878033
Epoch 11, Batch 1 loss: 4.842585
```

Optimizer: SGD

Learning Rate: 0.01

Batch Size: 32

Conclusion: Over fitting

Screenshot Shows Last Five Epochs

```
Epoch 16, Batch 1 loss: 0.050183
Epoch 16, Batch 101 loss: 0.069556
Epoch 16, Batch 201 loss: 0.103341
Epoch: 16       Training Loss: 0.102417         Validation Loss: 7.367409
Epoch 17, Batch 1 loss: 0.014588
Epoch 17, Batch 101 loss: 0.054086
Epoch 17, Batch 201 loss: 0.084003
Epoch: 17       Training Loss: 0.083942         Validation Loss: 7.166718
Epoch 18, Batch 1 loss: 0.009981
Epoch 18, Batch 101 loss: 0.049127
Epoch 18, Batch 201 loss: 0.068351
Epoch: 18       Training Loss: 0.068280         Validation Loss: 6.772996
Epoch 19, Batch 1 loss: 0.029676
Epoch 19, Batch 101 loss: 0.030793
Epoch 19, Batch 201 loss: 0.039253
Epoch: 19       Training Loss: 0.039517         Validation Loss: 7.022517
Epoch 20, Batch 1 loss: 0.002231
Epoch 20, Batch 101 loss: 0.016993
Epoch 20, Batch 201 loss: 0.026408
Epoch: 20       Training Loss: 0.037660         Validation Loss: 6.166826
```

Custom Neural Network Architecture

```
Net(
  (conv1): Conv2d(3, 32, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1))
  (conv3): Conv2d(64, 128, kernel_size=(5, 5), stride=(1, 1))
  (conv4): Conv2d(128, 256, kernel_size=(5, 5), stride=(1, 1))
  (conv5): Conv2d(256, 512, kernel_size=(5, 5), stride=(1, 1))
  (dropout): Dropout(p=0.4)
  (fc1): Linear(in_features=4608, out_features=500, bias=True)
  (fc2): Linear(in_features=500, out_features=133, bias=True)
)
```

i went on with 5 convolutional layers with kernel size of 5, stride and padding kept at their default values, 1 pooling layer that will be applied at each output of the convolutional layer to reduce the image size in half. I have also used dropout to make the model not over fit, i have tested with different values like 0.1,0.2,0.3,0.4,0.5 but found 0.4 to be sufficient for the job, as finally the validation loss was lesser in this case.

i have made 2 fully connected layers at the end of our cnns that will take the cnn output and produce 133 predicted class ids
activation functions:
i have applied relu activation function at every convolution layer output, and our fully connected layer except the last one.

Custom Neural Network Architecture (Transfer Learning)

I included resnet50 to use as part of transfer learning, as it is popular for image classification tasks, at the output of resnet50, I added a fully connected layer to make the output to our 133 classes. I have also freezed the values of our pretrained network and only trained parameters of our fully connected layer

I tried with vgg16 at first but was getting some errors, so I changed it

Results

Model evaluation and validation

Our both custom models with and without transfer learning are evaluated based on accuracy metrics. Below are some screenshots taken from last test performed on both networks

Custom Model successful test accuracy, the model was trained for 15 epochs, the training and validation was done 3 times for 5 epochs each and output loss was monitored. Below is screenshot of test accuracy function on our last successful test

```
Test Loss: 3.977399

Test Accuracy: 11% (100/836)
```

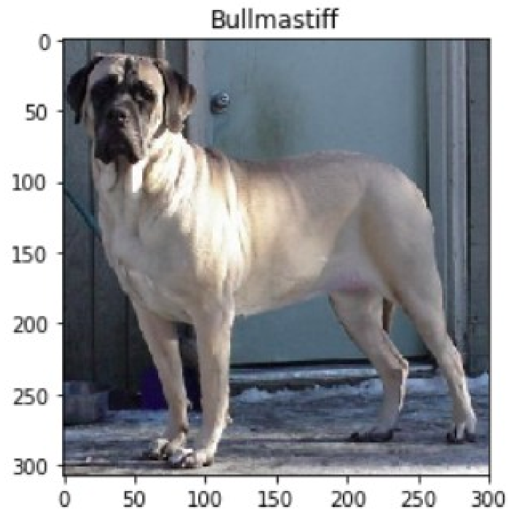
The model didn't perform well because our network is small and dataset is also small, later we can see the accuracy improve using transfer learning

The second model implementation of resnet50 with one fully connected trainable layer, was trained for 30 epochs. Same as in our custom model's case, I ran the epochs separately and monitored the output, and then decided to go for larger epoch. Below is screenshot of our test accuracy on the final model.

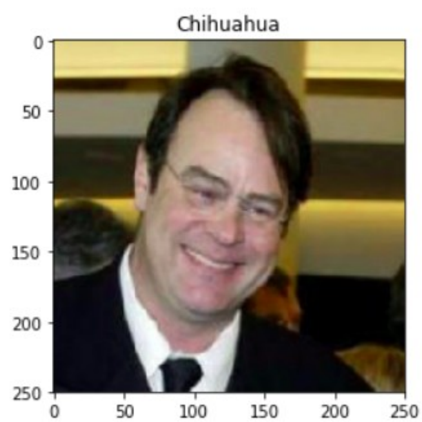
```
Test Loss: 1.522924

Test Accuracy: 77% (652/836)
```

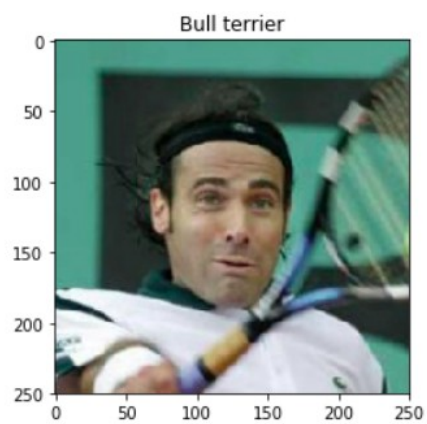
Final Trained Model Tests

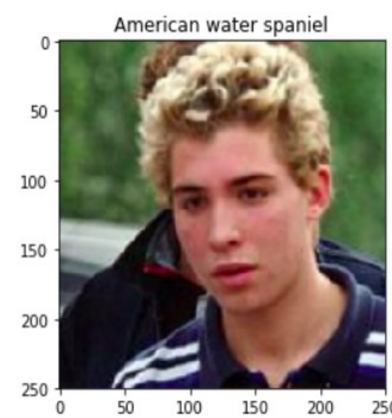
```
predict_breed_transfer(dog_files[1])
```

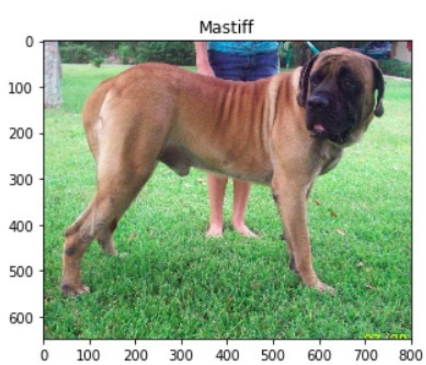


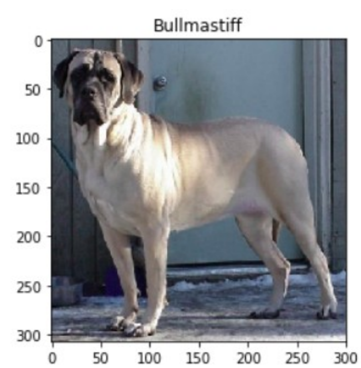

```
'Bullmastiff'
```

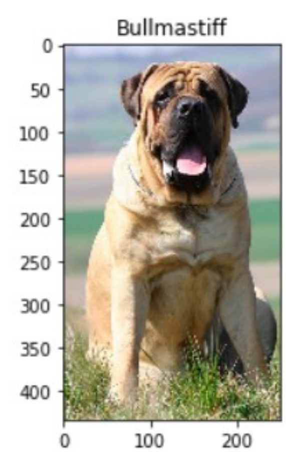












Conclusion

Reflection

The task involved required image classification, and a lot of training was involved, the task also required GPU to run successfully, I tried to run the project on my laptop with GPU but it still was running out of memory. I also tried to do it on amazon sagemaker but had issues. Finally I had to settle with udacity's provided workspace to complete the project. Coming back to the project, this was a wonderful project, and while doing the project I felt that the same task can be applied to any kind of object, these kind of classification tasks can be applied anywhere a person can imagine. The difficult part was defining the neural networks, choosing the optimal loss function and optimizer was challenging and time consuming. Other parts were quite easy as it was normal procedural work.

Improvement

Possible improvements are

- replace features of human face with predicted dog face, just for fun
- train the network for more epochs and its accuracy will improve
- try more pre-trained networks and see how they perform

References

Haar cascade OpenCV: , Haar feature-based cascade classifier, , https://docs.opencv.org/trunk/db/d28/tutorial_cascade_classifier.html
VGG-NET: , VGG-NETS, , https://pytorch.org/hub/pytorch_vision_vgg/
StackOverflow Reference: , StackOverflow Reference, , https://stackoverflow.com/questions/58151507/why-pytorch-officially-use-mean-0-485-0-456-0-406-and-std-0-229-0-224-0-2