Machine Learning Engineer Nanodegree

Capstone Project Report

Dog Breed Classifier

Han Chao Zhao

November 17, 2020

Definition

Project Overview

Convolutional neural network (CNN) is a deep learning algorithm that is well known for its effectiveness in analyzing images. Being able to identify a dog breed has been a popular problem in the machine learning space along with other type of classifiers like plant identification. For this project we will try to once again use CNN to solve the dog breed classifier problem.

Problem Statement

We want to be able to solve two problems for this project.

1) If a dog is detected, classify the breed of the dog.
2) If a human is detected, classify which dog breed the human resembles the most to.

To solve the problem above, we also need to be able to detect whether a given picture is a human, a dog, or neither.

Metrics

The evaluation metrics we use for this project is the test accuracy against the given dog dataset. For the CNN model built from scratch, the model should be able to achieve 10% accuracy. For the CNN model created using transfer learning, the model should be able to reach at least 60% accuracy.

Analysis

Data Exploration

Dog dataset

Training: 6680 images

Validation: 835 images

Test: 836 images

There are total 133 classes (breeds) of dogs in our dataset.

Human dataset

There are total 132233 human images that can be used to test the model at the end. These human images does not participate in training.

Exploratory Visualization

We mention that the task of assigning breed to dogs from images is considered exceptionally challenging. To see why, consider that *even a human* would have trouble distinguishing between a Brittany and a Welsh springer spaniel.

Figure 2: Welsh springer spaniel

Figure 1: Brittany

It is not difficult to find other dog breed pairs with minimal inter-class variation (for instance, Curly-Coated Retrievers and American Water Spaniels).

Figure 3: American Water Spaniels

Figure 4: Curly-Coated Retrievers

Likewise, recall that Labradors come in yellow, chocolate, and black. Your vision-based algorithm will have to conquer this high intra-class variation to determine how to classify all of these different shades as the same breed.

Figure 5: Yellow Labrador

Figure 6: Chocolate Labrador

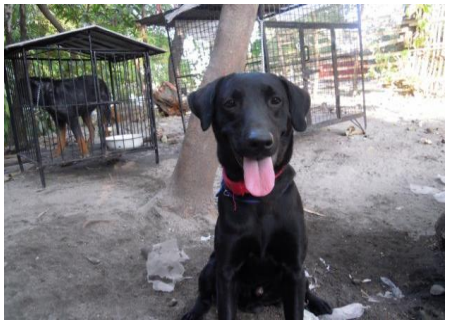


Figure 7: Black Labrador

We also mention that random chance presents an exceptionally low bar: setting aside the fact that the classes are slightly imbalanced, a random guess will provide a correct answer roughly 1 in 133 times, which corresponds to an accuracy of less than 1%.

Algorithms and Techniques

For human detection, we use OpenCV's implementation of Haar feature-based cascade classifiers. It is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001.

For dog detection, we use a pre-trained VGG-16 model.

Then we will create a CNN model from scratch, we would mimic a basic architecture setup that is used in the Udacity CNN course, and modifying it as we proceed to achieve better results.

Next, we will pick a pre-defined architecture as our CNN model and use it to achineve better results for our classification test.

Benchmark

For benchmarking, we are going to use random guessing as the benchmark and hope to create a CNN model to surpass it. Given the 133 types of dog breed, the accuracy of random guessing is 1/133 around 0.7519%

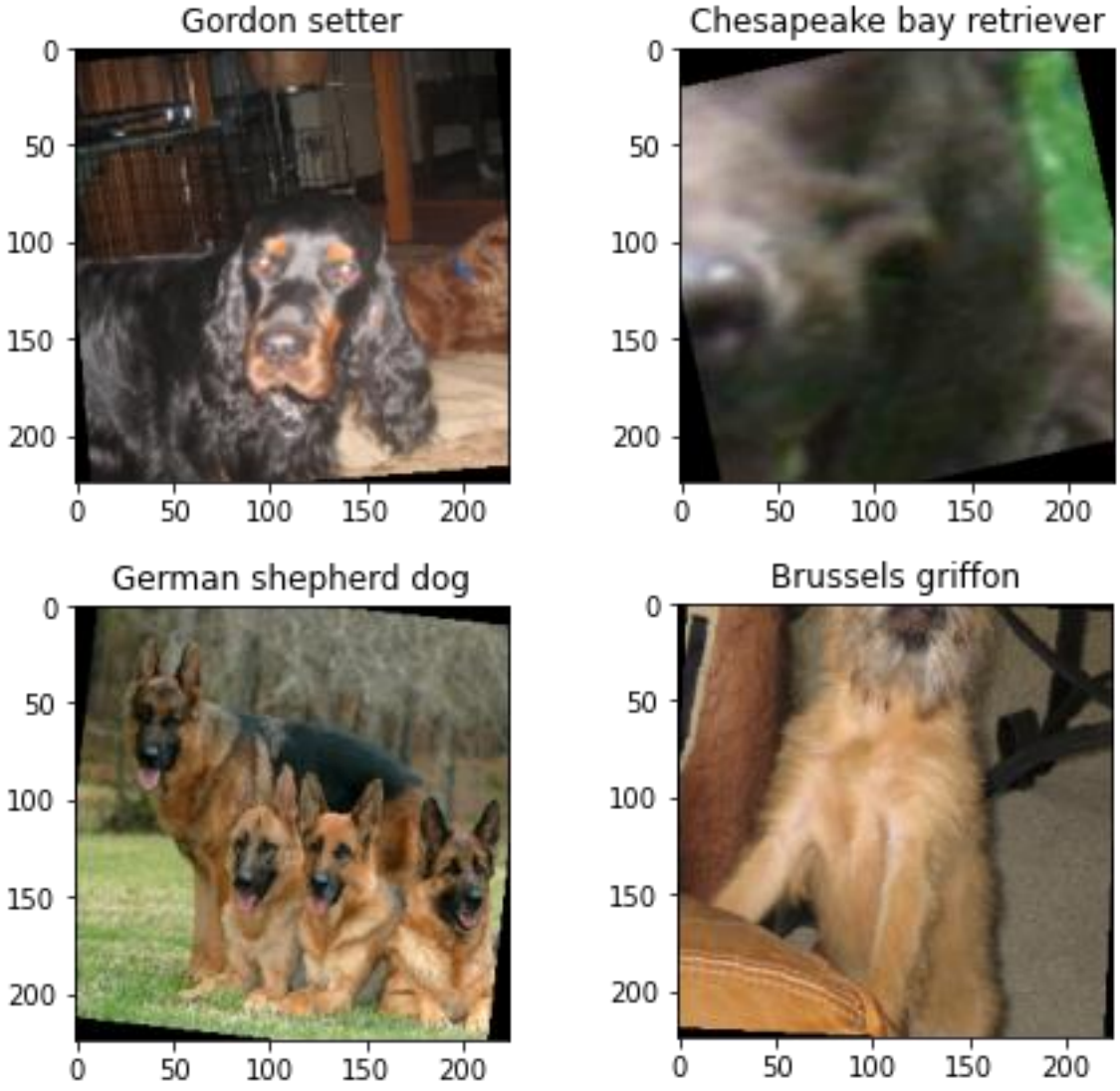
Methodology

Data Preprocessing

For each the training dataset for dogs, we apply the following preprocessing steps to each picture.

- A random resized crop to make the picture the size of 224x224.
- A random horizontal flip.
- A random rotation of maximum 15 degrees.
- A normalization of mean=[0.485, 0.456, 0.406, std=[0.229,0.224, 0.225]])

Here are some of the example images after preprocessing:



For validation and test datasets we are only applying the random resized crop and normalization.

Some of the data augmentation were added as the effort to increase the accuracy of our model.

Implementation

The custom CNN model consist of the following layers.

```
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))
  (fc1): Linear(in_features=12544, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=133, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (batch_norm): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
```

The forward behavior is defined as the following:

```python
def forward(self, x):
    ## Define forward behavior

    # sequence of conv and max pooling layer
    x = self.pool(F.relu(self.conv1(x)))
    x = self.pool(F.relu(self.conv2(x)))
    x = self.pool(F.relu(self.conv3(x)))
    x = self.pool(F.relu(self.conv4(x)))
    x = self.pool(F.relu(self.conv5(x)))

    x = x.view(-1, 7*7*256)

    x = F.relu(self.batch_norm(self.fc1(x)))
    x = self.dropout(x)

    x = self.fc2(x)
    return x
```

1) conv1 sees 224x224x3 tensor and convert it to 224x224x16 with the max pooling layer to change it to 112x112x16
2) conv2 sees 112x112x16 tensor and convert it to 112x112x32 with the max pooling layer to change it to 56x56x32
3) conv3 sees 56x56x32 tensor and convert it to 56x56x64 with the max pooling layer to change it to 28x28x64
4) conv4 sees 28x28x64 tensor and convert it to 28x28x128 with the max pooling layer to change it to 14x14x128
5) conv5 sees 14x14x128 tensor and convert it to 14x14x256 with the max pooling layer to change it to 7x7x256
6) A dropout layer of 0.20 is used to prevent over fitting.
7) Two layers of linear transformation that will get the number of output to 133.

Refinement

During development, we made a few changes to the CNN model to achieve better accuracy by adding more convolution layers, adding more data augmentations, modifying the learning rate, and experimenting with the dropout value.

The ultimate refinement is when we use transfer learning to create a CNN model, we picked Resnet50 as our final CNN architecture as it was the winner of ImageNet challenge in 2015. It has reasonable amount of convolution layers so the training speed is within reason.

Results

Model Evaluation and Validation

For Haar feature-based cascade classifiers, for the 100 human and dog pictures we tested on, 96% of the human pictures were correctly identified as human. 18% of the dog pictures were wrongly identified as human.

For the pre-trained VGG-16 model, for the 100 human and dog pictures we tested on, 95% of the dog pictures are identified as dog. No human pictures were identified as dog.

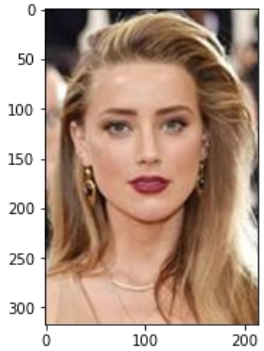
The CNN model built from scratch achieved 19% accuracy by correctly identifying 164 dog pictures' breed from the 836 test images.

The CNN model created using transfer learning achieved 74% accuracy by correctly identifying 620 out of 836 dog pictures correctly.

Justification

The accuracy of the final CNN model exceeds the project's requirement and it is a huge improvement from random guessing and the CNN model built from scratch. Here are some example of the tests.

Looks like you are a human!

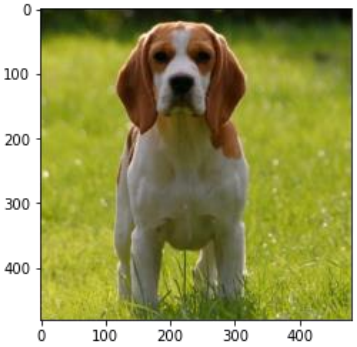


You look like a: Cavalier king charles spaniel

Looks like you are a human!

You look like a: Welsh springer spaniel

Looks like this is a dog!



It might be a: American staffordshire terrier

Looks like this is a dog!

It might be a: Plott