Machine Learning Nanodegree Capstone Report

Cooper Chastain

March 12, 2021

Definition

Project Overview

The goal of this project was to develop an algorithm that could estimate the breed of a dog from an image of a dog or, if a human is detected, estimate the breed that most resembles the human.

This project was performed using a Jupyter notebook provided by Udacity that simulated a project development pipeline. The pipeline included loading and preprocessing images, developing and testing various detection algorithms that would be used in the final algorithm, and finally developing the final algorithm.

The overall success of the project depended on two metrics, the accuracy of a from scratch dog breed detection algorithm and the accuracy of a transfer learning dog breed detection algorithm. The project satisfied both.

The final algorithm was subjected to a very small set of never before seen, real world images and performed reasonable well on them.

The review for the project proposal may be found here.

Problem Statement

Develop an algorithm that can estimate the breed of a dog from a user supplied image. If the image is of a human, the algorithm will provide an estimate of the dog breed that is most resembling.

The algorithm should be able to correctly identify the breed in at least 60% of images where a single dog and no human is present.

Evaluation Metrics

The primary metric was be accuracy, but other metrics such as precision, recall, sensitivity, and specificity, and F1 score were be calculated.

Analysis

Data Exploration

There are 8,351 dog images divided into 133 classes and 13,233 human images.

Key observations:

- In most cases the images do not have an aspect ratio of 1 (i.e. they are not square)
- Some images contain more than one dog and others contained humans and dogs.

Although it is likely that the dog breed images were imbalanced, this was not explicitly determined because it was not clear, at the time of project development, how this information could be used.

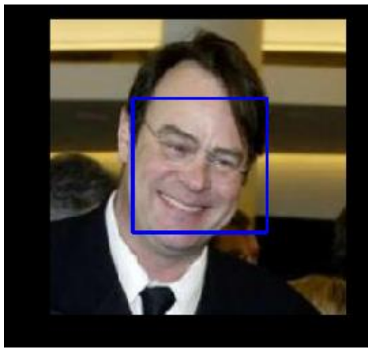
Exploratory Visualization

The only exploratory visualization that was performed was a visual inspection of some of the images. An example of a dog and human image is shown below.

Algorithms and Techniques

1) Face Detection

   A "Face Detector" algorithm is used to detect the presence of humans. This algorithm uses Haar-cascade Detection in OpenCV to not only detect a human face but to also wrap the face in a bounding box as shown below.



The “Face Detector” takes the output of the cascade classifier and checks to see how many faces were detected. If the result is greater than 0, it reports True.

2) Dog Detection

   A “Dog Detector” algorithm is used to detect the presence of a dog. This algorithm uses a pre-trained and unmodified version of VGG16. This version of VGG16 will detect more than just dogs, however, so the “Dog Detector” evaluates the result of the VGG16 model to see if the output falls into the range 151 and 268 (inclusive) which corresponds to various dog breeds. If it does, it reports True.

   It takes a 224x224 image as an input.

3) Estimate Dog Breed: from scratch model

   The first attempt to build a dog breed detection algorithm involved the use of a “from scratch” model. For this project, a multilayer CNN was chosen.

   The original model was inspired by the winner of the CIFAR competition a few years ago but then simplified.

   The initial model was inspired by the following:

   - A desire to use more than 3 layers of CNNs to see what would happen
   - An attempt to use a repeating pattern of CNNs, for example, CNN -> MP -> Drop in a way similar to the CIFAR winner. It was later found that removing the dropouts between the CNNs improved results.
   - A desire to use increasing levels of dropout like the CIFAR winner. Once dropouts between the CNNs were removed this became unnecessary
   - A desire to get the width and height down to a low number at the input of the flattening function, like 4 or in this case 7
   - The use of default padding and strides
   - A 2 layer hidden layer since the number of CNNs was greater than 3.

   The initial model did not work as well as expected and was simplified as follows:

   - 6 layers of CNN reduced to 5, each with stride and padding equal to 1
   - 2x2 Max Pooling between layers, but no Dropout
   - 30% Dropout between the final CNN and the Flattening layer
   - A 2 layer hidden layer with 20% Dropout between layers

   An input size of 224x224 was chosen to match that of the VGG input.

   With an input size of 224x224 and the 5 CNN layers, the output of the CNN layers is a 7x7x256 tensor.

4) Estimate Dog Breed: Transfer Learning with VGG16

   To improve upon the performance of the from scratch model, a transfer learning approach was taken. The VGG 16 model was chosen as the model because it was used earlier for the "Dog Detector" algorithm, it is a proven algorithm for dog detection, and because the author was already familiar with it from other projects (e.g., AI Programming with Python Nanodegree).

   Other algorithms were considered, such as AlexNet and resnet. The author compared these algorithms with VGG16 during the AI Programming with Python Nanodegree and found that VGG16 outperformed them when testing for dog versus cat detection. The results are shown below.

Model	pct_match	pct_correct_dogs	pct_correct_breeds	pct_correct_notdogs
AlexNet	75	100	80	100
VGG16	87.5	100	93.33	100
resnet18	82.5	100	90	90

   The VGG16 model was chosen over other VGG16 models because the author was already familiar with it and because the author believed that the more complicated VGG models, such as VGG19, would not provide any material improvement over VGG16.

   The VGG16 model was modified by replacing the very last hidden layer (i.e., the output layer) by reducing the number of outputs (i.e. classes) to 133 to match the number of dog breeds. A decision was made not replace the entire hidden layer because it was reasoned that the earlier stages of the hidden layer were well suited for dog breed detection (it was already doing it, along with detecting other objects). Further, the only layer that was optimized during training was the last layer.

Benchmark

The benchmark model is the VGG16. This model was used in the AI Programming with Python Nanodegree program to determine if an image was of a dog or cat.

In this project, VGG16 was used in both unmodified and a modified form.

The target metric for dog detection using the from scratch model was 10% accuracy and for the transfer learning model 60%.

Methodology

Data Preprocessing

Dog images were already separated into three sets: train, test, and validation.

All images are scaled to 256x256 and then cropped to 224x224. The reason for this is to 1) zoom into the image a little bit to enlarge the detectable object (e.g., the dog or human) and 2) size the image for input into the two models.

All images are converted to tensors and normalized.

Training images are subject to data augmentation in the form of random crop, random horizontal flip, and random rotation. Random crop has the effect of shifting the image around whereas horizontal flip and random rotation change the orientation.

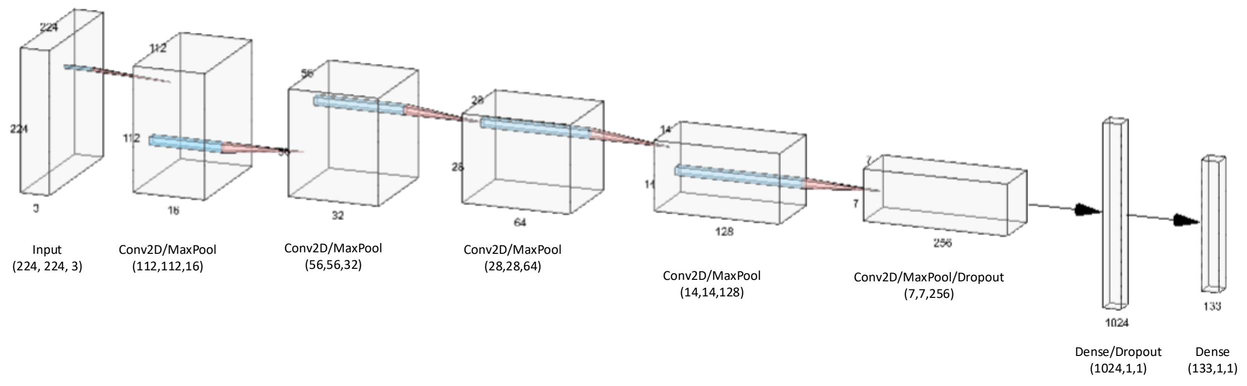
Validation and Test images are center cropped after resizing.

Implementation

Dog images were already split into three groups: train, valid, and test. Each set was transformed using `transforms.Compose` and loaded using `datasets.ImageFolder`. The same transformers were used for both the from scratch and transfer learning models.

Samplers for train and valid were created using `SubsetRandomSampler`. Batch image loaders for all three were created using `torch.utils.data.DataLoader`. The loaders were used for both the from scratch and transfer learning models.

The from scratch model was created by defining as a class in `class Net`. Strides and Padding are set to 1 and MaxPool set to 2. Dropout between the last CNN layer and the first dense layer is set to .3 and between the dense layers set to .2. A VGG style rendition of the model is shown below.



For both models, `nn.CrossEntropyLoss()` was used as the loss function and `optim.SGD()` with the learning rate set to 0.05 as the optimizer. The learning rate was set to 0.05 because at the original setting of 0.01, up to 100 epochs (with a 3 hour running time) were required to train the models.

Models were trained through a model training function. The function calculated both the training loss and the validation loss. Models were saved when the validation loss improved.

The model training function can be turned off using the global constants `TRAIN_SCRATCH_MODEL` and `TRAIN_TRANSFER_MODEL`. Setting these constants to `False` allows the notebook to be run without performing training.

The final algorithm takes as inputs the results of "Face Detector," "Dog Detector," and "Breed Detector" and applies simple boolean logic.

- If both a dog and human are detected, the result is both a human and a dog are detected. The detected dog might be the result from "Breed Detector."
- Else, if only a human is detected, the result is a human and the dog breed that most resembles the human is the result from "Breed Detector."
- Else, if a dog is detected but not a human, the result is a dog and the dog breed that it most like is from "Breed Detector."
- Else (finally), if neither dog nor human are detected, then an error is given.

Refinement

The from scratch model was refined several times to improve its results. The refinements involved updates (and in some cases corrections) to data preprocessing as well as changes and simplifications to the original model. The steps taken to refine the model improved the accuracy score from 13% to 18%.

It was noted that augmenting the data improved the accuracy score.

It was further noted that applying the same transforms, minus augmentation, to all three sets increased the accuracy score.

It was noted that removing Dropout between the CNN layers improved the accuracy score.

It was noted that using a common input size of 224x224 allowed for greater reuse of code.

The transfer learning model did not require refinement as its accuracy was 84%.

<u>**Results**</u>

<u>Model Evaluation and Validation</u>

The from scratch dog detection algorithm achieved an accuracy of 18% which is higher than the 10% minimum criteria.

The model with VGG16 and Transfer Learning achieved an accuracy of over 84% which is far above the minimum criteria of 60%

From a visual inspection, the final algorithm was pretty good with the never before seen images from the internet.

<u>Justification</u>

It is quite apparent that both the from scratch and transfer learning models surpassed the minimum criteria for dog breed estimation.

Further, the results of the final algorithm using never before seen images from the internet were quite good. They were not 100% accurate, but they were good enough.