Final Report: Dog Breed Image Classification with CNN

Jason Zeng

July 7, 2020

Project Overview

This project aims to identify the breed of dog. Given an image of a dog, our algorithm will identify an estimate of the canine's breed. If supplied an image of a human, the algorithm will identify the resembling dog breed. The idea is to build a pipeline to process real-world, user-supplied images. This is a multi-class classification problem and we can use supervised learning to tackle it.

Domain Background

This project is derived from Computer Vision.

Problem Statement

The goal of this project is to build a machine learning model that can be used as a web app to process real-word, user-supplied images. It requires the implementation of two kinds of detections. First, **Dog detector**, which can identify the dog's breed when an image of a dog is given. Second, **Human detector**, which will identify the resembling dog breed if an image of a human is supplied.

Datasets and Inputs

The dataset for this project is provided by Udacity. They are pictures of dogs and humans. To be more specific, **Dog images dataset** has 8351 images in total which are split into train (6,680 images), test (836 images) and valid (835 images) directories. Each of this directory has 133 folders corresponding to dog breeds. These images have different sizes, different quality, and are imbalanced for different breeds. On contrary, **Human images dataset** has 13233 images in total and are stored in 5750 folders based on human names. Though these human images have different backgrounds and are also imbalanced, they are of the same size 250x250 pixels.

Sample images from input

Solution Statement

We can use Convolutional Neural Networks (CNN) for this kind of multi-class classification tasks, by feeding images into the algorithm. The CNN architecture then learns knowledge (weights w and bias b) by minimizing the difference between CNN's prediction with provided ground truth (labels). This minimization process is done through back propagation.

In this project, we will first use existing algorithms like OpenCV's implementation of Haar feature based cascade classifiers, to detect human images. Next, we will use a pretrained VGG16 model to detect dog images. Finally, after the image is identified as dog or human, we can pass this image to an CNN to predict the breed out of the 133 possible breeds.

Benchmark Model

- The CNN model created from scratch should have accuracy at least ~10% since a random guess has accuracy < 1% (1 out of 133 breeds).
- The CNN model created using transfer learning should have a much higher accuracy, say 70%.

Evaluation Metrics

For multi-class classification, multi-class log loss is a typical metric to be used to evaluate model performance. Accuracy is also used to get a feeling of overall model performance.

Data Preprocessing

This includes train/test/validation split and image augmentations on training. To be more specific, all images are resized (to 224x224) and normalized. The training images are randomly rotated and randomly flipped horizontally. All images are converted into tensor before passing into the model.

Benchmark model from scratch

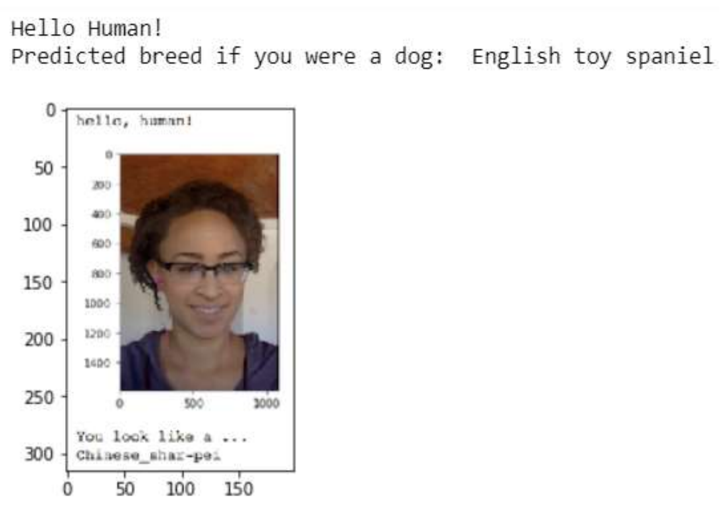
Building from scratch, a 3-layer CNN architecture is built. The first conv layer takes input images of size 214x214 and the final conv layer outputs a size of 128. Two fully connected layers produce a final output with dimension 133 (133 breeds). ReLU activation function is used. A 2x2 pooling layer is also used to reduce input size by a factor of 2, and a dropout of 0.25 is applied to reduce possible overfitting.

This benchmark model built from scratch has an accuracy of 11%, which is significantly better than random guess (1/133~1%) but is still poor for our application.

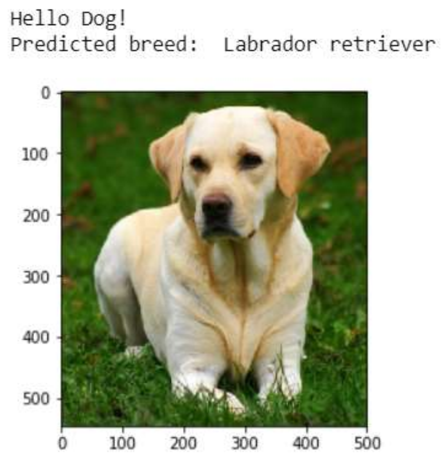
Improved model with Transfer Learning

I created a CNN architecture using Transfer Learning with ResNet101. The benefit of ResNet has been extensively discussed in literature. Overall, it is guaranteed to learn at least an identify operation and sometimes can learn a little bit more by focusing on the residuals, thus leading to improvements. Another benefit of ResNet is it won't suffer from vanishing gradient like conventional deep neural networks.

By utilizing ResNet101 architecture, the last conv output of Resnet101 is fed as input to our model. A fully connected layer is added to produce 133-dimensional output, with one for each dog breed. The model performs very well that it got 75% accuracy within just 5 training epochs.

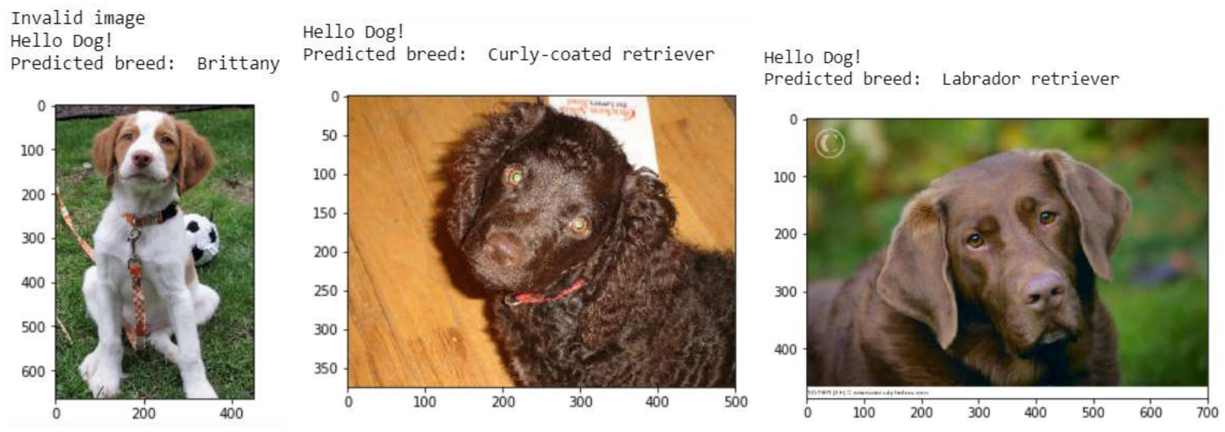


```
Hello Human!
Predicted breed if you were a dog:  English toy spaniel
```



```
Hello Dog!
Predicted breed:  Labrador retriever
```

Sample prediction output from the improved model with transfer learning



```
Invalid image
Hello Dog!
Predicted breed:  Brittany
```

```
Hello Dog!
Predicted breed:  Curly-coated retriever
```

```
Hello Dog!
Predicted breed:  Labrador retriever
```

Sample prediction output from the improved model with transfer learning

Model Evaluation and Validation

Human face detector: This detector uses OpenCV's implementation of Haar feature based cascade classifiers. In the first 100 images of human face dataset, 98% of human faces were detected; in the first 100 images of dog face dataset, 17% of human faces were detected.

Dog face detector: This detector uses pre-trained VGG16 model. In the first 100 dog images, 100% of dog faces were detected; in the first 100 human images, 1% of dog faces were detected.

CNN with transfer learning: A CNN model using transfer learning with ResNet101 architecture was built. After 5 training epochs, the model produced an accuracy of 75% (630/836) on test data. The CNN model created using transfer learning gives much better accuracy than the CNN model we built from scratch.

Future Improvements

To further improve the model, we can try these things:

- Add more data would help improve training and get better models. This includes adding more images on dogs / human or use more image augmentations or a mixture of both.
- Perform hyper-parameter tuning. This includes learning rate, drop-outs, batch sizes, optimizers, loss function etc..
- If possible, additional information from other sources may be included to further improve prediction accuracy.
- Try different architecture

References

1. Original project repo on GitHub from Udacity:
   https://github.com/udacity/deep-learning-v2-pytorch/tree/master/project-dog-classification
2. Resnet101:
   https://github.com/KaimingHe/deep-residual-networks
3. Pytorch:
   https://pytorch.org/