Machine Learning Capstone – Project Report

1. Definition

1.1 Problem statement

The goal of this project is to create an artificial intelligence model that can classify images of dogs based on their breed, with at least 60% accuracy. For fun, this A.I. will also identify human faces in an image, and output which dog breed the human face most closely resembles.

1.2 Project overview

This project involves elements of image processing, computer vision, data science, and machine learning. At its core, the project is an image classification problem, but a quite complex problem, since differentiating between different dog breeds based on purely visual characteristics alone is not a trivial problem. Even human observers can sometimes have trouble differentiating between certain similar looking dog breeds.

Solving the problem of dog breed classification, using cutting edge deep learning technology is therefore a useful study in 1) harnessing the power of artificial intelligence to achieve human like or even super human accuracy and 2) demonstrate end-to-end learning.

Within this project, I will perform the following series of tasks:

1. Import the required datasets

2. Create a function to detect dogs in an image

3. Create a function to detect human faces

4. Create a convolutional neural network (CNN) from scratch to classify dog breeds

5. Create a CNN using transfer learning, to get a much more accurate dog breed classifier

6. Write an algorithm that first identifies whether an image is that of a dog or a human. If it is a dog, the algorithm classifies that dog according to breed. If it is a human face, the algorithm says which dog breed the human most closely resembles.

7. Test this algorithm out on sample images.

The data used for this project comes from Udacity's Dog Classification Project GitHub repository. The dataset consists of 13233 images of human faces, and 831 images of dogs. The dog images are divided into a training set (80% of the images), a validation set (10% of the images), and a test set (10% of the images). There are 133 classes of dog breed to train the classifier on. Each category in the dataset has approximately 25 to 60 images. This is a relatively balanced dataset, but to really ensure a fair sampling of training images, I will experiment with using data augmentation, with PyTorch's built in augmentation functionality.

1.3 Metrics

The only metric used here will be classification accuracy. This is because this particular dataset is relatively balanced when it comes to number of images per class, which means a high classification accuracy on the test set is a good enough indicator of how well this algorithm performs as a dog breed classifier. I will aim for an accuracy of at least 10% with the CNN built from scratch and at least 70% on the CNN built with transfer learning.

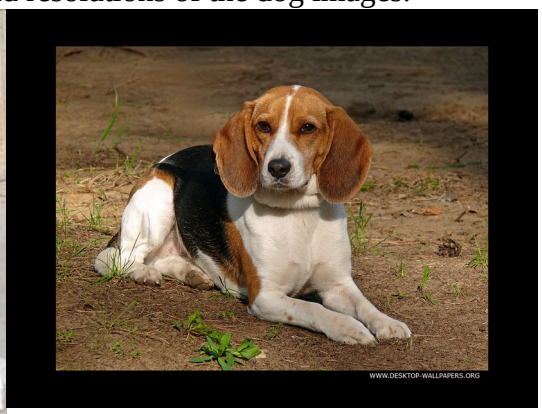
2. Analysis

2.1 Data exploration

As mentioned before, there are 831 images of dogs and 13233 images of human faces in the dataset. The dogs are divided into 133 different classes based on breed. Preliminary exploration of the data shows images of various sizes and resolutions. There are roughly 20 to 60 images per category in the dataset. This leads me to believe that it will be advantageous to use some sort of image augmentation while training the models. This will increase the diversity of data being fed into the model and hence help it generalize more to unseen test data.

2.2 Exploratory visualization

Some of the images from the dataset are shown below. As is apparent, there is a fair amount of variety in the postures, facing angles, and resolutions of the dog images.



2.3 Algorithms and techniques

In a nutshell, the techniques I used in this project were image preprocessing and augmentation using PyTorch's built in transformers, building a CNN from scratch using PyTorch, and experimenting with transfer learning using PyTorch and its built in pre-trained networks, trained on the ImageNet dataset.

As this problem is an image classification problem, using deep convolutional networks and transfer learning seemed to be the best possible approach. Deep learning has shown to be extremely promising with image data classification. Using a pre-trained network with transfer learning helps, because pre-trained convolutional networks have already detected useful features and combinations of features pertaining to images in the ImageNet dataset, that could be useful for classifying dog breeds. After all, there are several categories of dog breeds in the ImageNet dataset itself, so these pre-trained networks already have the ability to identify dog-breed specific features.

2.4 Benchmarks

In this project, I first built a neural network classifier from scratch and then built a more accurate one using transfer learning. Looking at the results of the neural network built from scratch, I was able to get an accuracy of about 13% on my test data. I took this as a benchmark, and built another model with transfer learning to see hoe much of an improvement over the from-scratch network I could achieve.

3. Methodology

3.1 Data preprocessing

The first step in classifying the image data is preprocessing the images. First, all the images had to be the same height and width (224 x 224), irrespective of whether they came from the training, validation, or test dataset, because this is the input size expected by the VGG16 architecture, and also for the architecture I plan to build from scratch. Secondly, all images had to be normalized the same way.

Since each category of dog breed in the dataset has about 20 to 60 images, I decided a bit of data augmentation might help the model with generalization. So, for the training data loader, I preprocess the data by randomly resizing and cropping a 224 x 224 sized window. I then add a transform to randomly flip the image horizontally with a probability of 0.5. Finally I add a transform to randomly rotate the images between +10 and -10 degrees. These transforms should ensure a greater diversity of image data per class, and hence should help the model generalize better to unseen data while training. I do not add these transforms to the validation and test data loader as it this data is not used to train the network and hence adding diversity information here is unnecessary.

3.2 Implementation

For the CNN model that I built from scratch to classify dog breeds, I started of by using the same architecture used in the Cifar10 dataset classification example in the Udacity Machine Learning Nanodegree extracurricular notes. This architecture used 3 convolutional layers, each with filters of

size 3x3 and same padding. Each of these 3 convolutional layers was followed by a 2x2 max pooling layer with stride 2, effectively halving the output size of the previous layer. After the convolutional layers, the output was flattened and sent through two fully connected layers to the output prediction layer. This architecture also made use of dropout at the fully connected layers, with a dropout probaility of 25%. The outputs of every convolutional layer, as well as every fully connected layer except the last, were fed through a RELu activation before being propagated to the next layer.

I believed this would be a good enough starting point. However on second thoughts, I added another convolutional layer and another fully connected layer, since the task of classifiying dog breeds is more complex, and hence would require both a deeper feature extraction pipeline as well as a more complex classifier function.

I chose the cross entropy loss here, because this ensures that the output of the neural network is converted into a probability using a softmax operation, and then converted into a log probability and used to caclulate the loss, all in a single line of code. I also chose the ADAM optimizer instead of a vanilla gradient descent optimizer. ADAM has shown to be a superior optimizer in several computer vision applications and challenges. It is faster and more robust to local minima in the loss curve.

3.3 Refinement

Later, I built a model using transfer learning. I spent quite some time on this part of the project. The first thing I tried was to use the VGG16 network for transfer learning. I replaced the final, fully-connected layer of the VGG16 network with a new fully connected layer with 133 output nodes, froze the weights of all layers except for the last fully connected layer, and then trained this new network on the dog dataset. I was negatively surprised with the results of this. I wasn't able to get an accuracy beyond 40%, with this network. I am still not sure why the pre-trained VGG16 network did not perform well with transfer learning, but if I had to guess, I would say it is because the fully connected layers of the pre-trained VGG model were "overfit" in away to the original ImageNet dataset, and couldn't be fine-tuned to the new dog breeds dataset in a reasonable number of iterations. I suppose one way of overcoming this would be to replace the entire fully connected part of the pre-trained VGG16 model, or at least more of the fully connected layers than just the last layer. However, I decided instead to try my luck with a different pre-trained network first.

I chose the ResNet50 next, for several reasons. First, ResNet50 achieved a higher accuracy on the original ImageNet dataset than VGG16 (about 93.3% vs 91.9%). This would imply that ResNet50 has a better feature exraction pipeline than VGG16. Secondly, ResNet50 has an architecture that comprises of residual connections and many deep layers, compared to VGG16's straightforward convolutional + fully connected architecture. This gives ResNet50 an advantage of strengthening feature propagation from the original image data to the later layers. With these general thoughts in mind, I tried transfer learning with the pre-trained ResNet50 model, simply replacing the last fully connected layer with my own, and training the weights on just this layer. This was enough to give me an accuracy of 83% in 10 epochs.

4. Results

4.1 Model evaluation and validation

The final test accuracy that I obtained with the transfer-learning based model was 83%. I then used this model to build my dog detector algorithm and ran it on my own sample images. The model did reasonably okay on these images, identifying dogs and humans accurately, and classifying the dogs according to their breed (or classifying them into similar looking breeds).

4.2 Justification

The model output is as good as I aimed for at the beginning of this project. However, it can potentially be improved a lot further. Some possible ways of improving it would be:

1. Training the transfer-learned model for more epochs. There was a downward trend of training and validation loss when I stopped the training, so I assume I could further improve the accuracy just by training longer.

2. Using a larger dataset / more data augmentation. There are many web-scraping libraries I could use to get hundreds more images of various dog breeds from Google searches, and add this to my data library. Also, augmentation techniques such as tweaking the image brightness, saturation and hue, or random rotations, or random grayscaling etc would result in training a network that is far better at generalizing to unseen test data.

3. Doing some research on different pre-trained networks to use as a starting point. I looked at the VGG16 network and ResNet50, but there are several others, such as ResNet152, Inceptionv3 etc, that could potentially give better results.

4. More hyperparameter tuning. Maybe using an adaptive learning rate, or some kind of learning rate scheduler. Maybe using a different optimizer than Adam (Nadam for instance seems quite promising). Or even adding more layers to the end of the network, with different activation functions etc. There is a lot of trial and error based optimization that can be done here.