Udacity Machine Learning Nanodegree

Capstone Project: Dog Breed Classifier

George Xian Wee

Project Definition

Overview

This project originates from one of the capstone proposals provided by Udacity in it's Machine Learning Engineer Nanodegree. It is a dog breed classifier which integrates several components to create a simple application to identify dogs and humans and to classify the type of dog breed, or the type of dog breed a human most closely resembles. The project falls under the research domain of image recognition, a field in machine learning where major breakthroughs have been achieved, such as a convolutional neural network (CNN) winning 2012 ImageNet Large Scale Recognition challenge. The training, validation and test dataset in this project was provided by Udacity. Additional test images have been included as well.

Problem Statement

The project primarily addresses classification problems using supervised learning. Input images are mostly of dogs and human faces but the application can actually accept any image. The first classification problem is to determine if an image contains a dog (the model will classify an image as dog or not dog). If there is a dog in the image, a convolutional neural network will take the dog image as an input and output a prediction of the breed of the dog. If there is not a dog in the image, the application will detect if there is a human face in the image and if one has been found, the CNN will guess which kind of dog breed the human face most closely resembles. In the case where an image does not contain a dog or human, the application will display a message noting this state.

Metrics

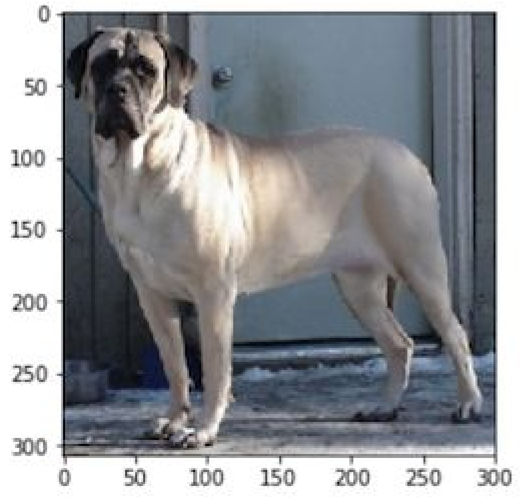
In this project accuracy (number of correct predictions / total number of predictions) is used as the primary metric for evaluation and is suitable because the data is fairly balanced, meaning the training, validation and test data have roughly the same number of samples of dogs of each breed. A balanced dataset is important to avoid misleading accuracy results, such as as case in which 90% of the test data images are of a specific breed - then a default prediction of that breed would yield 90% accuracy.

Analysis

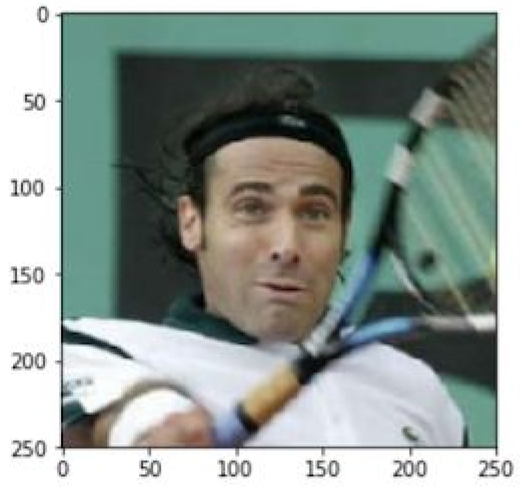
Data Exploration and Visualization

The dataset provided by Udacity contains 8351 dog images and 13233 human images. Each dog and human image in its raw form has R G B values and may vary in its pixel dimensions. For example, an image tensor could have the following dimensions: [1, 3, 224, 224] - 1 image, a list RGB values of length 3 and width and length of 224.

Sample dog image



Sample human image



Methodology

Data Preprocessing

The images were retrieved using the Python Image Library (PIL) and transformed to 224 x 224 pixels, converted to tensors and normalized. After the image transformations, a sample dog or human image would have dimensions 1 x 3 x 224 x

224, meaning the first tensor would contain 3 tensors representing RGB values of a 224 x 224 image. Each pixel is an input into the initial layer of the neural network.

The images were normalized with mean values [0.485, 0.456, 0.406] and standard deviation values [0.229, 0.224, 0.225]. Since PyTorch's ResNet model was used as the starting point for transfer learning, I used the recommended mean and standard deviation values from PyTorch's ResNet documentation (https://pytorch.org/hub/pytorch_vision_resnet/). Images were transformed into uniform dimensions by first resizing them to 256 pixels and then center cropping to 224 pixels, again similar to the process and dimensions used to train PyTorch’s ResNet model.

Implementation and Refinement

First I defined, trained, validated and tested a convolutional neural network from scratch. The CNN contains 2 layers of convolutions which produce features used by a final, fully connected layer that takes in the features and outputs predictions. The first convolutional layer has 3 input channels to account for the 3 dimensions of R, G, B values, 6 output channels, a kernel size of 5 for each sliding filter and a stride of 1. Generally, odd numbers are chosen for kernel sizes in the convolutional layer to avoid distortions that occur across layers with even numbered kernel sizes. For example, common kernel sizes in convolutional layers are 3x3, 5x5 and 7x7. Smaller strides tend to encode more information and maintain translational invariance, meaning the effect of the position of the object in the image is reduced. The second convolutional layer has 6 input channels which is the same number of output channels as from layer 1, 16 output channels and a kernel size of 5.

To downsample, or to reduce the dimensionality of each layer, max pooling was used to obtain the maximum value from the activated features in the previous layer (the maximum activation). A kernel size and stride of 2 were chosen, which is consistent with common practice. A small kernel size and stride in pooling prevents discarding too much information from the previous layer.

Finally, fully connected layers take in the features from the convolutional layers and produces predictions. The input layer has 16 * 5 * 5 parameters (the number of outputs from the last convolutional layer multiplied by the dimension of the filters in the 2nd convolutional layer). The hidden layers consist of 120 and 84 neurons. There are a few empirically driven rules of thumb when choosing the number of hidden neurons. In general, the number of hidden neurons should be between the number of input and output parameters. Too few hidden neurons may result in underfitting and too many hidden neurons may lead to overfitting. A common rule is to have the number of hidden neurons be roughly the average of the number of neurons in the input and output layers. The output layer contains 133 neurons which corresponds to the 133 dog breeds in our dataset. Each neuron of the output layer is a probability assigned to each dog breed, with the highest probability representing the model's prediction of the type of dog breed in the image.

The prediction results of a neural network built from scratch and trained over 25 epochs of 64 images per batch was quite low, only about 1% accuracy (10/836). To improve the performance, I employed a form of transfer learning to extract the features from a pre-trained neural network. I started with a pre-trained neural network, ResNet18, a residual neural network, which is a convolutional neural network containing "skip connections" to simplify the network by using fewer layers for training and to avoid "vanishing gradients" problem in which the gradients in the early layers of a neural network become extremely small, inhibiting the network's ability to accurately reflect how a small change in a parameter's value will affect the output.

Next I replaced the final prediction layer of the network with a fully connected layer consisting of 512 input features (the output from the final convolutional layer of the model) and 133 output nodes (the number of dog breeds in our dataset). The final layer was re-trained with Cross Entropy Loss criterion and stochastic gradient descent optimizer to update the parameters with a learning rate (how much to adjust each parameter based on the gradient) of .001 and a momentum (the accumulation of previous gradients to determine how much to update the parameters) of .9, a common value used in practice. Learning rates (typically ranging from .1 - .001) that are larger enable models to train faster but may lead to suboptimal final weights while smaller learning rates train more slowly but may lead to better results. The result, discussed in the next section, substantially improved.

Results

Model Evaluation and Validation

To evaluate both the model created from scratch and the transfer learning model, I used the following validation process. I accumulated the loss during each forward pass and computed the average validation loss by dividing the total loss in each epoch by the number number of samples in the validation set. The model was validated on 835 images. To track the progress of the training, I accumulated the training loss in each epoch and calculated the average training loss. The model was trained on 6680 images.

Justification

For the convolutional network built and trained from scratch, the results were poor. After training for 25 epochs, the training loss only decreased from 4.893726 to 4.876308. Validation loss was reduced from 4.892617 to 4.877803. The results of the model on the test set was similar, a 4.871027 loss. The model was only able to achieve an accuracy of about 1% (10/836).

I then experimented with increasing the learning rate by an order of magnitude, to .01, to see if performance would improve. Over a course of 25 epochs, training loss was reduced to 4.254043 from 4.854727. Validation loss dropped to its lowest point in epoch 23, to 4.258644. Performance appeared to stall at this point as validation loss increased in epoch 24 to 4.273353 and in epoch 25, to 4.314453. The accuracy achieved by a model trained on a higher learning rate improved from 1% to about 4.5%.

The results from transfer learning were substantially better. In epoch 1, the training loss started at 4.617218 and by epoch 25, was reduced to 0.639790. The validation loss began at 4.095313 and ended at 0.586960. When the trained transfer model was tested on the test dataset, the model's loss was 0.798060 and achieved an accuracy of 81% (684/836).

Demo of Application

```python
def run_app(img_path):
    """Application
    Args:
        Img_path (str): path to image

    Returns:
        None: Outputs result of predictions
    """

    is_dog = detect_dog(img_path)

    if is_dog:
        print('Dog detected!')
        dog_breed = predict_dog_breed(img_path)
        class_names = [item[4:].replace("_", " ") for item in image_datasets['train'].classes]
        idx_to_class = {idx: breed for idx, breed in enumerate(class_names)}
        print('Dog breed: ', idx_to_class[dog_breed])
        display_image(img_path)
    else:
        is_human = detect_human(img_path)

        if is_human:
            print('Human detected!')
            display_image(img_path)
            dog_breed_resemblance = predict_dog_breed(img_path)
            class_names = [item[4:].replace("_", " ") for item in image_datasets['train'].classes]
            idx_to_class = {idx: breed for idx, breed in enumerate(class_names)}
            dog_breed_resembled = idx_to_class[dog_breed_resemblance]
            file_paths = [file for file in dog_files if dog_breed_resembled.replace(' ', '_') in file]
            if file_paths:
                print('Dog breed resembled: ', dog_breed_resembled)
                display_image(file_paths[0])
            else:
                print('No image path found')
        else:
            print('Not a dog or human')
```

Example 1: Dog Detected

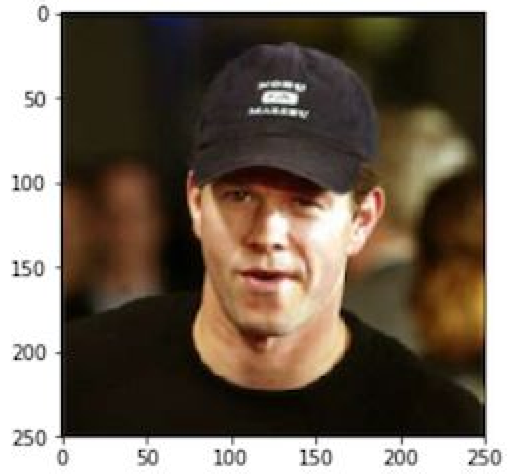
```
Dog detected!
Dog breed:  Dachshund
```

ground truth: /data/dog_images/test/056.Dachshund/Dachshund_04003.jpg

Example 2: Human Detected

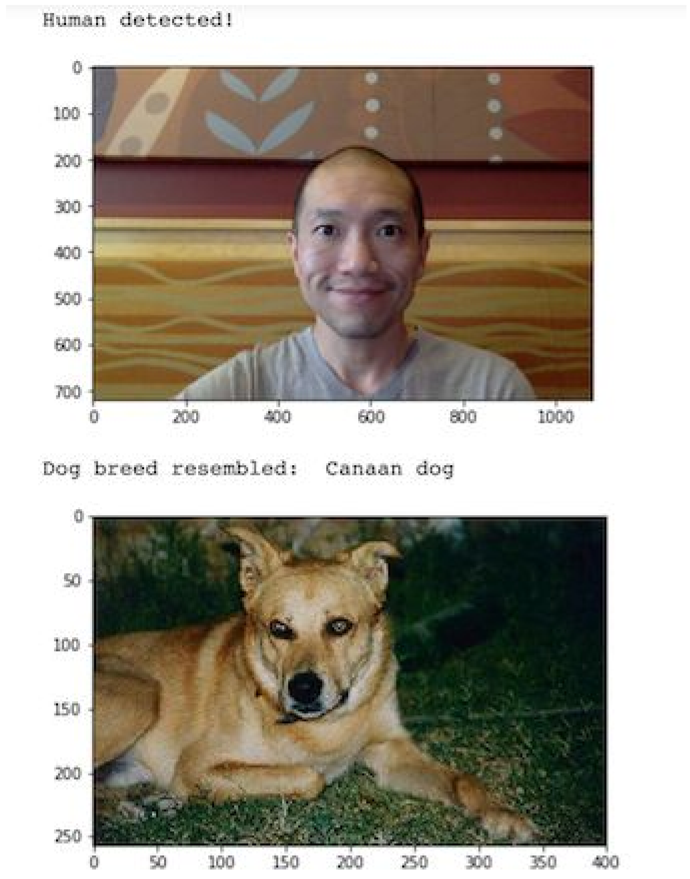
Person: /data/lfw/Mark_Wahlberg/Mark_Wahlberg_0001.jpg

```
Human detected!
```



```
Dog breed resembled:  Doberman pinscher
```

Example 3: Human Detected


```
Human detected!
```
```
Dog breed resembled:  Canaan dog
```

Conclusion

In this project, images of dogs were preprocessed, a convolutional neural network was trained, validated and tested, and a function was written to simulate an application that takes in an input image path, predict the breed of the dog or the type of breed a human most closely resembles.

Building a convolutional neural network that can achieve high accuracy from scratch was tougher than I expected. Initially I thought that by increasing the learning rate, I could increase the accuracy, which did occur but only up to a point. To improve the results of the model trained from scratch, I can try adding more convolutional layers and to boost the accuracy of the transfer model, perhaps I can experiment with adding more hidden layers to the final fully connected layers of the network.