Machine Learning Engineer Nanodegree

Report for Capstone Project: Dog Breed Classifier with CNN

Zhiguang Xue

October 6, 2021

Definition

Project Overview

Given an image of a dog, a dog breed classifier can give an estimate of the dog breed. It can be helpful in the scenario that a dog owner is unsure about the breed of his dog, and he wants to confirm it with someone or a classification web app by simply showing the picture. The work for building the dog breed classifier belongs to image classification, which is a research topic in computer vision.

Computer vision is one of the main technologies that enables the digital world to interact with the physical world, and it enables self-driving cars to make sense of their surroundings. Most computer vision algorithms use a convolution neural network, or CNN. A CNN is a model used in machine learning to extract features, like texture and edges, from spatial data, which is different from a normal feedforward neural network, where the input image would be flattened into a feature vector. A deep CNN model can achieve good prediction accuracy for image classification given a huge amount of training data, such as the ImageNet database.

In this project, a CNN model was constructed for classifying the dog breeds. The datasets for training the CNN model were provided by Udacity, containing 8351 dog images and 13233 human images. The entire dog dataset has 133 breed classes from Affenpinscher to Yorkshire Terrier, which means that a random guess of the dog breed will provide a correct answer roughly 1 in 133 times, corresponding to an accuracy of less than 1%.

Problem Statement

In this project, we will learn how to build a pipeline to process real-world, user-supplied images. Given an image of a dog, the designed algorithm/app will identify an estimate of the canine's breed. If supplied an image of a human, the algorithm will identify the resembling dog breed. If the image contains neither a dog nor a human, the algorithm should provide output indicating this. The prediction accuracy should not be too low, and we need to attain at least 60% accuracy. To achieve this goal, transfer learning is used, where we freeze the weights for all the network except the final fully connected layer.

In summary, the following are what we need to achieve:

- The algorithm can tell if the image contains a dog or a human, or neither a dog nor a human
- In the presence of a dog or a human, the algorithm can give an estimate
- The accuracy of the estimate should be at least 60%
- To achieve the high accuracy, we can build a CNN model using transfer learning

Metrics

I will use accuracy as the evaluation metric. Accuracy is a common metric and is easy to understand. It is suited for binary as well as a multiclass classification problem which is balanced and not skewed. In the later data exploratory visualization, we can observe slight class imbalance in the dataset, but this slight imbalance should not affect the effectiveness of using accuracy as the evaluation metric. We can get the accuracy of the classification model by counting the number of correct predictions for the dog images in the test dataset.

Analysis

Data Exploration

The datasets are provided by Udacity, with a dog dataset[1] containing 8351 dog images and a human dataset[2] containing 13233 human images. Since I will be using Udacity workspace for this project, and the datasets are already stored in the /data folder, I do not need to re-download the datasets from amazon S3 bucket. The human dataset can be used for detecting a human face and for testing the performance of the dog breed classifier. The dog dataset will be used for training the classifier. The dog images are under /data/dog_images with 3 categories:

- 6680 images under /data/dog_images/train for training
- 835 images under /data/dog_images/valid for validation
- 836 images under /data/dog_images/test for testing

The entire dataset contains 133 breed classes from Affenpinscher to Yorkshire Terrier.

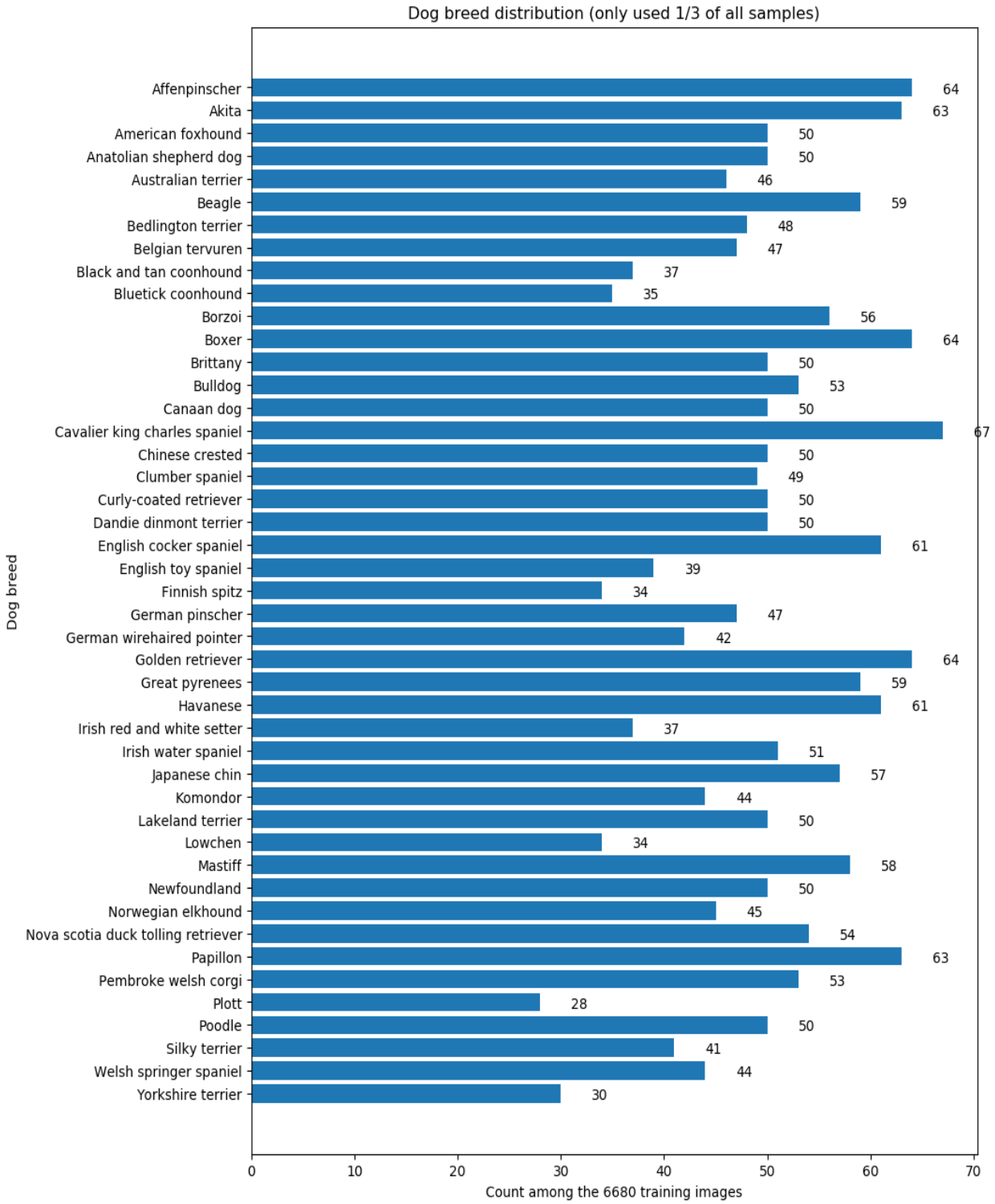
Exploratory Visualization

---

[1] Udacity. *The dog dataset.* URL: https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip

[2] Udacity. *The human dataset.* URL: https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip

The training data in the dog dataset contains 133 dog breeds and 6680 images. The following visualization shows the distribution of 1/3 dog breeds. The 1/3 dog breeds were selected with no bias by selecting the first breed of every 3 dog breeds, using python code index%3==0.



From the above visualization, we can see that the distribution of different dog breeds is slightly unbalanced, but this slight imbalance should not cause a problem. The average number of images that each dog breed relates to is about 50.

Algorithms and Techniques

Based on the problems that we need to address in this project, I will design a human face detector and a dog detector to tell if an image contains a human or a dog. Then I will use transfer learning to design a CNN model, which will be compared with a benchmark model that is built from scratch. Following are the specific parts:

1. **A human face detector**
   OpenCV[3] provides many pre-trained face detectors, stored as XML files on github. We can download one of these detectors and stored it locally for detecting human faces in a sample image. The human face detector is designed using OpenCV's implementation of Haar feature-based cascade classifier.

2. **A dog detector**
   The dog detector will be designed using pre-trained VGG-16 model, which was trained on ImageNet, a very large and popular dataset used for image classification and other vision tasks. VGG-16 model can be accessed by using the machine learning framework PyTorch[4]. As a dog detector, the loaded VGG-16 model is only used for dog breed prediction and its weights would not be updated. If the predicted index of VGG-16 model is between 151 and 268, a dog is detected.

3. **A CNN to classify dog breeds from scratch**
   This model is used as a benchmark model, and more details will be discussed in the next section. Since the input dataset is not large, I will use PyTorch to design a relatively simple CNN architecture consisting of 4 convolution and pooling layers, and 2 fully connected layers.

4. **A CNN to classify dog breeds using transfer learning**
   Because we want to have a dog breed classifier that has at least 60% prediction accuracy, transfer learning can help with only small amount of training dataset. Pre-trained ResNet-50 model will be loaded for transfer learning for its good top-

[3] OpenCV. URL: https://opencv.org/
[4] PyTorch. URL: https://pytorch.org/

1 accuracy (76.130) and top-5 accuracy (92.862)[5]. Its last fully connected layer will be replaced with a new one having output dimension 133. I will freeze the weights of all the network except the last fully connected layer.

5. **A dog breed classifier**
   In this part, I will put everything together for the dog breed classifier. First, use the dog detector and the human face detector to tell if the image contains a dog or a human. If neither a dog nor a human is present, an error message will be returned. If yes, then we will use the CNN model from transfer learning to get a prediction. The algorithm will perform as expected.

Benchmark

In this project, I will build a CNN model from scratch as the benchmark model, because I want to know how much the transfer learning can improve the prediction accuracy. Considering the small amount of the training dataset, the architecture of the CNN model from scratch will be relatively simple, consisting of 4 convolution layers and 2 fully connected layers. I expect that the test accuracy of this benchmark model is at least 10%.

Methodology

Data Preprocessing

From the previous data exploratory visualization section, we know that the distribution of the dog dataset among different dog breeds has a very slight imbalance, and this imbalance will not cause a problem.

Then what I will focus on is applying data augmentation, normalization, and resizing on the input images for the training/testing for both model from scratch and model using transfer learning. The model using transfer learning is based on the pre-trained ResNet-50 model, which requires the input images having a 224x224 pixels. For convenience, I also designed the model from scratch taking 224x224 dimension images as input. The data transformation was performed with PyTorch module torchvision.transforms.

For the 6680 training dog images, I first cropped a random portion of an image and then resized it to the 224x224 pixels by bilinear interpolation. I have also augmented the training dataset by random crop and random horizontal flip. For 835 validation and 836 test images, I first resized the

[5] ResNet-50 model accuracy scores were found on PyTorch website. URL: https://pytorch.org/vision/stable/models.html

image to 256x256 pixels, and then center cropped the image to 224x224 pixels. After resizing, the normalization was applied.

Implementation

In the section of algorithms and techniques, I listed 5 components that we will implement. Here, I would explain their implementations in more details.

1. **A human face detector**
   The human face detector is designed using OpenCV's implementation of Haar feature-based cascade classifier. For a given image, we first convert the image to grayscale, and then apply the pre-trained classifier. Each detected face is a 1D array with four entries that specifies the bounding box of the detected face. If the size of the array is not zero, at least one face is detected.

   We tested the performance of the face detector by 100 human images and 100 dog images. The face detector successfully detected 98 human images and only missed 2. But it detected 17 images from 100 dog images, which is higher than the expected percentage. Based on this fact, in the final implementation of the dog breed classifier, I would first apply a dog detector, and then apply the human face detector.

2. **A dog detector**
   The dog detector was designed using pre-trained VGG-16 model, which was accessed by using PyTorch. Given an image, the VGG-16 model returns a prediction (derived from the 1000 possible categories in ImageNet[6]) for the object that is contained in the image. The prediction index corresponding to dogs appear in an uninterrupted sequence and correspond to values 151-268, inclusive, to include all categories from 'Chihuahua' to 'Mexican hairless'. Thus, in order to check if an image is predicted to contain a dog by the pre-trained VGG-16 model, we need only check if the pre-trained model predicts an index between 151 and 268 (inclusive). Also, VGG-16 takes 224x224 pixel image as input, so we need to resize and normalize an input image.

   We also tested the performance of the dog detector by 100 human images and 100 dog images. The result is ideal, 100% of dog images with a detected dog and 0% of human images with a detected dog. So, in the final app of dog breed classifier, I would use dog detector to check the input image before using the human face detector.

[6] ImageNet. URL: https://www.image-net.org/

3. **A CNN to classify dog breeds from scratch**
   This model is used as a benchmark model, and I used PyTorch to design, train and test the model from scratch. Because the training dataset is not large, I have designed a relatively simple CNN architecture: 4 convolution + pooling layers, and 2 fully connected layers. As a common practice, the number of channels in the later convolution layers have been increasing, and the height and width of the image in the later convolution layers have been decreasing by applying max pooling. A dropout layer with probability 0.3 is applied after the first fully connected layer to prevent overfitting. Following are the architecture:

   - 1st Convolution (3 input channels, 6 output channels, and kernel size 5) + Relu activation + Max pooling (kernel size 2)
   - 2nd Convolution (6 input channels, 16 output channels, and kernel size 5) + Relu activation + Max pooling (kernel size 2)
   - 3rd Convolution (16 input channels, 32 output channels, and kernel size 4) + Relu activation + Max pooling (kernel size 2)
   - 4th Convolution (32 input channels, 64 output channels, and kernel size 4) + Relu activation + Max pooling (kernel size 2)
   - Flatten all dimensions except batch
   - 1st Fully connected layer (7744, 1024) + Relu activation + Dropout (p=0.3)
   - 2nd Fully connected layer (1024, 133) as output layer

4. **A CNN to classify dog breeds using transfer learning**
   Pre-trained ResNet-50 model was loaded with PyTorch for transfer learning for its good top-1 accuracy (76.130) and top-5 accuracy (92.862). I freezed the weights of all the network except the last fully connected layer by setting requires_grad = False. The last fully connected layer in ResNet-50 (2048, 1000) was replaced with a new one (2048, 133) with random weights, and only this layer was trained.

   ```
   (fc): Linear(in_features=2048, out_features=133, bias=True)
   ```

5. **A dog breed classifier**
   In this section, I put everything together for the dog breed classifier. First, use the dog detector and the human face detector to tell if the image contains a dog or a human. If neither a dog nor a human is present, an error message is returned. If yes, then I use the CNN model from transfer learning to get a prediction. Please note that the input image needs to be resized to 224x224 pixels and then normalized before being passed to the CNN predictor.

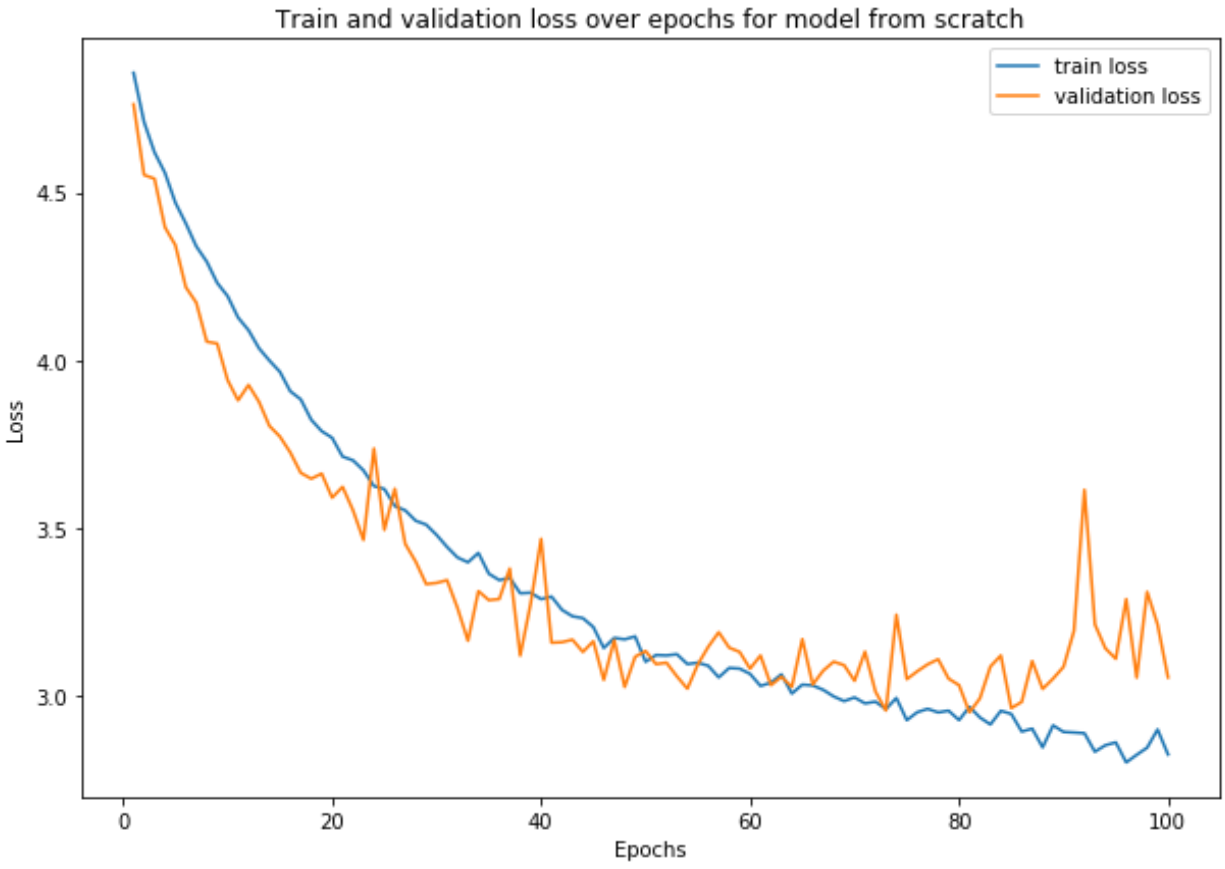
Refinement

The architecture of the CNN model, the pre-trained model for transfer learning, the optimizer, number of epochs, data augmentation, and the loss function will all affect the prediction accuracy. I have tested different architectures for the model from scratch, and tested different epoch numbers for the model using transfer learning. After refinement, the accuracy only improved a little.

Results

Model Evaluation and Validation

During the training of both models, the model was validated with the validation dataset after each epoch. The model with the smallest validation loss was saved. After the training, the models were tested with the test dataset, and their prediction accuracy were calculated.

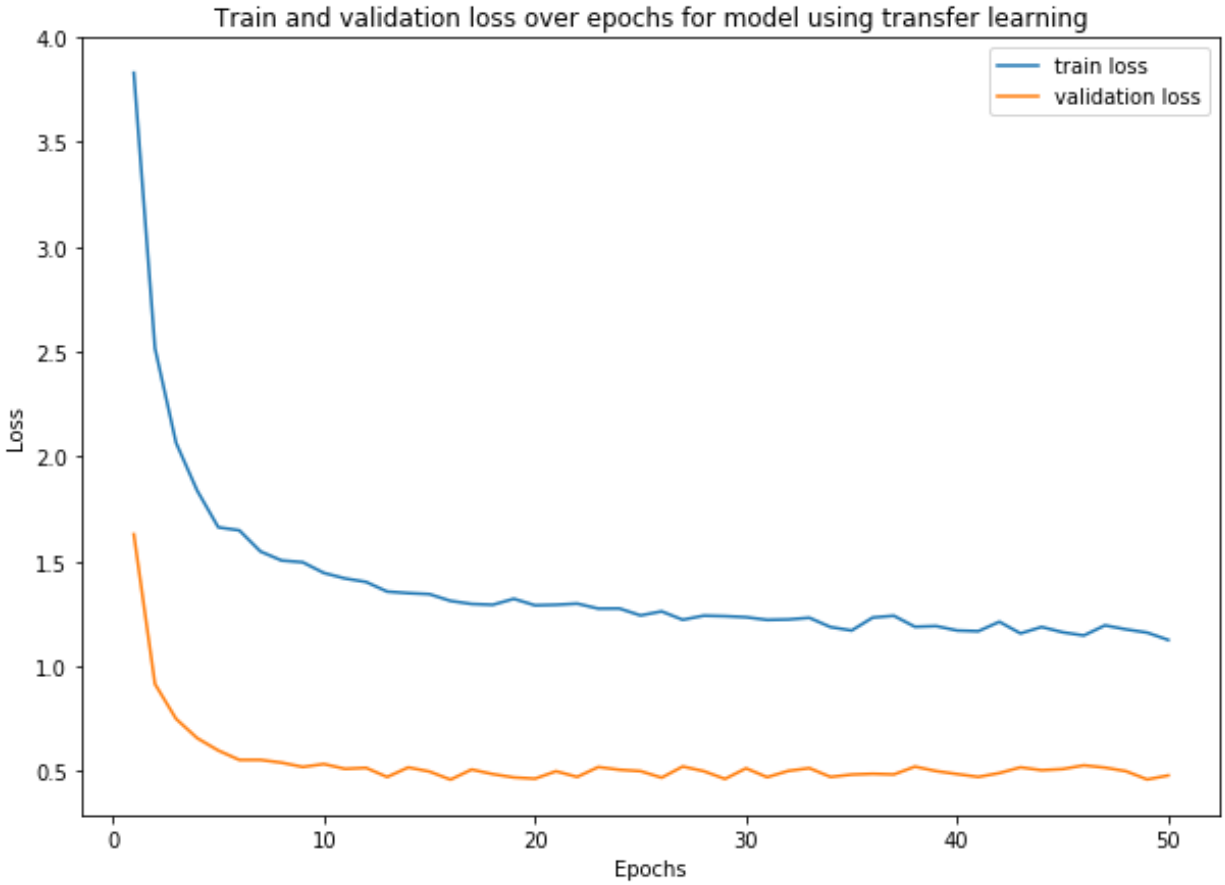


For **training model from scratch**, 1) number of epochs was set to 100, which was enough for this training; 2) cross entropy loss function was used by calling CrossEntropyLoss() function in PyTorch; 3) for optimizer, I selected stochastic gradient descent (SGD) optimizer with learning

rate 0.001 and momentum value 0.9, because SGD works very well for image classification. The above figure shows the evolution of the training loss and the validation loss over the epochs during training.

From the figure, we can see that a good convergence was probably reached around 60 epochs, and after that, the validation loss tended to increase even though the training loss continued to decrease. But I did not worry the overfitting issue, since the model with the smallest validation loss was saved.

For **training model using transfer learning**, number of epochs was set to 50 instead of 100, because 50 was enough. Similarly, cross entropy loss function was used by calling CrossEntropyLoss() function in Pytorch; SGD optimizer with learning rate 0.001 and momentum value 0.9 was used by calling SGD() function. The following figure shows the evolution of the training loss and the validation loss over the epochs during training:



From the above figure, we can see that after 20 epochs, the training loss decreased very slowly, and the validation loss almost did not change after 10 epochs. Therefore, 20 epochs should be enough, and the current setting 50 epochs would not harm.

Justification

Both models were tested with 836 testing images, and following are the accuracy scores:

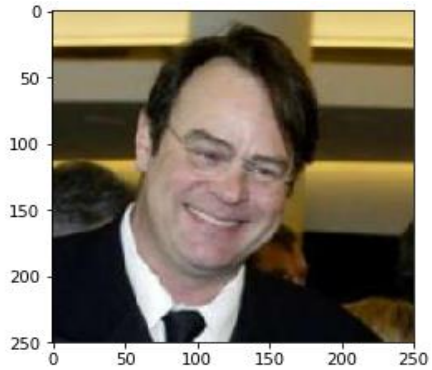
- 27% for model from scratch (benchmark model)
- 86% for model using transfer learning

Both models have achieved the pre-defined accuracy threshold: 10% for benchmark model, and 60% for model using transfer learning. The model using transfer learning was used for the dog breed classifier, since 86% accuracy was enough to address the problem of this project.

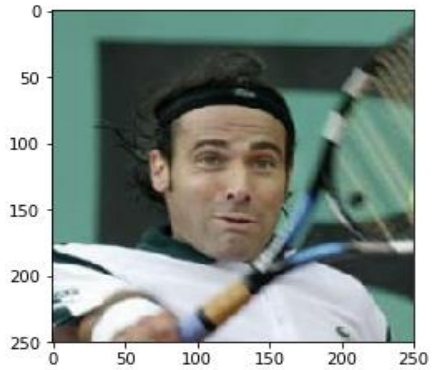
Conclusion

Free-Form Visualization

Here are several testing results:

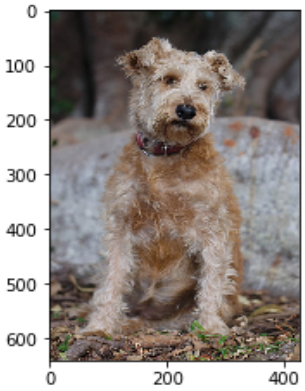



```
Hello human!
If you were a dog, you look like a ...
Chihuahua
Image file name for checking:  /data/lfw/Dan_Ackroyd/Dan_Ackroyd_0001.jpg
```



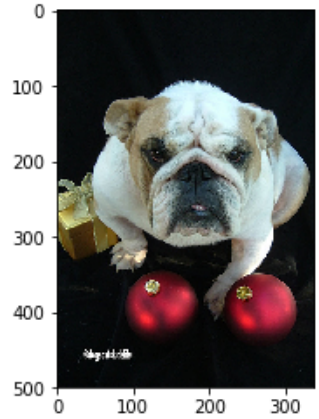

```
Hello human!
If you were a dog, you look like a ...
American foxhound
Image file name for checking:  /data/lfw/Alex_Corretja/Alex_Corretja_0001.jpg
```

```
Hello dog!
Your predicted breed is ...
Manchester terrier
Image file name for checking:  /data/dog_images/train/102.Manchester_terrier/Manchester_terrier_06779.jpg
```



```
Hello dog!
Your predicted breed is ...
Irish terrier
Image file name for checking:  /data/dog_images/test/087.Irish_terrier/Irish_terrier_05895.jpg
```

```
Hello dog!
Your predicted breed is ...
Cane corso
Image file name for checking:  /data/dog_images/valid/044.Cane_corso/Cane_corso_03176.jpg
```




```
Hello dog!
Your predicted breed is ...
Bulldog
Image file name for checking:  /data/dog_images/train/040.Bulldog/Bulldog_02823.jpg
```

Reflection

The process used for this project can be summarized using the following steps:

1. Import datasets
2. Design a human face detector
3. Design a dog detector
4. Preprocess the data, and divide the data into test/validation/test datasets
5. Create a CNN model to classify dog breeds from scratch (benchmark model)
6. Create a CNN model to classify dog breeds using transfer learning
7. Combine the two detectors and CNN model using transfer learning to form the app of dog breed classifier
8. Test the app of dog breed classifier with random images

I found the step 6 the most difficult, so I had to familiarize myself with the transfer learning process using PyTorch.

Improvement

Here are three possible points for algorithm improvement:

- Turn the algorithm into a web application for better user accessibility
- Collect more data or augment the training data more to improve the model performance
- Try more accurate pretrained models (like ResNet-152) for transfer learning to improve the prediction