Capstone Report

Chu Gong

2020-08-23

I choose Dog Breed Classifier(CNN) as my capstone project.

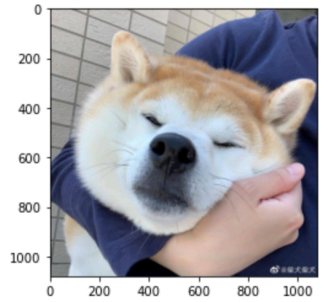
1 Definition

1.1 Porject Overview

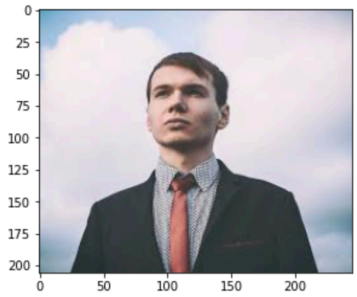
Computer Vision is a popular field that have derived many interesting academic researches and industry applications. CV related techniques are aimed to tackle problems such as human face recognition and object detection for automatic cars. As the develpoment of artificial intelligence and deep learning algorithms within these years, people can solve these problems with higher accuracy and efficiency. For example, from the simple CNN architectures like LeNet-5 and AlexNet, to deeper model such as VGG and GoogLeNet, these models help people achieve better performance on image classification tasks[1]. In order to get a closer look at how CNN works and build my own interesting image classification app, I choose this capstone project and hope to explore more advanced techniques of deep learning.

In this project, we built an deep learning model based app to infer which breed the dog is likly to belong to. If the app figured out that the input image is a dog, it will return the prediction of dog breed. If the app figured out that the input iamge is a human, it will also return a result that which dog breed is most similiar to this human. From this intersting project, I learned that CNN model and those popular architecture are powerful for those object detection and classification problem, and can be further implemented in more problems in real world.

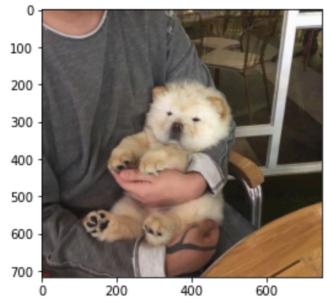
The dataset we use is *dogImages* dataset of different breeds of dog and its labels. Also, we can use our own images to play with this app, and the final result looks like this:



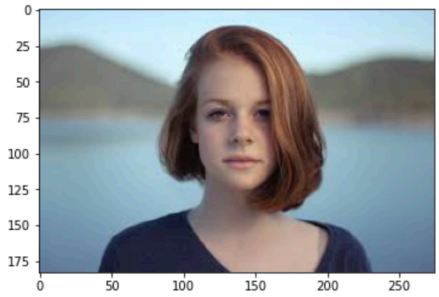
hi dog, you probably is Akita



hi human, you looks like American water spaniel



hi dog, you probably is Chow chow



hi human, you looks like Maltese

1.2 Problem Statement

The problem of this project is to build an app and relevant algorithms to detect dogs in the image and identify the breed of the dogs. For a given image that user inputs, my model are expected to output the most possible answer of the dog's breed. And if the image is not a dog but a human, my model can detect the human and give a resembling breed as output[2]. The basic model I plan to use is CNN(convolutional neural network) but the detailed architecture of cnn model need to be explored and specified. In order to measure and decide which model is more suitable for our dog recignition problem, I plan to compare the accuracy of predictions on test set.

We need several solutions for different steps of this project. First task is to detect people and dogs in the image. In this step, we use OpenCV's pre-trained face detector and VGG-16 network to solve this problem. These pre-trained models can be implemented by extracting xml files directly from github or other open source platform. The second task is to classify dog breeds and similar dog breeds for people. We can design our own cnn model architecture from scratch or use transfer learning based on some pre-trained architecture such as *Inception or ResNet*. After specified the model architecture, the following pipeline is same for each machine learning model, from optimizer definition, model training to validation on test set. Finally I will compare the test set classification error and decide the best performance model to use.

1.3 Metrics

I choose *accuracy* as the evaluation metric of this project. In face detection section, we will test our detector model on 100 images of human files and 100 images of dogs files. So the accuracy can be defined as *the number of faces that model detected / 100* for each detector. In the breeds classification section, *accuracy = number of correct classification / total number of test set* .

Also, since this is a classification problem and accuracy metric may cause some issues when the dataset is imbalance, so we can also use confusion matrix and F1 score to improve our metrics. Besides, since our problem is a multi-classification, we can calculated F1-score for each class of dog breeds and use weighted average all classes as our final matric. However in this problem, we have total 133 classes of breed and each class has around 26-58 training images. The number of class is relatively large and the potential sample imbalance may not cause much influence over all classes. So accuracy is enough for us to compare the perforamance over different models.

2 Analysis

2.1 Data Exploration

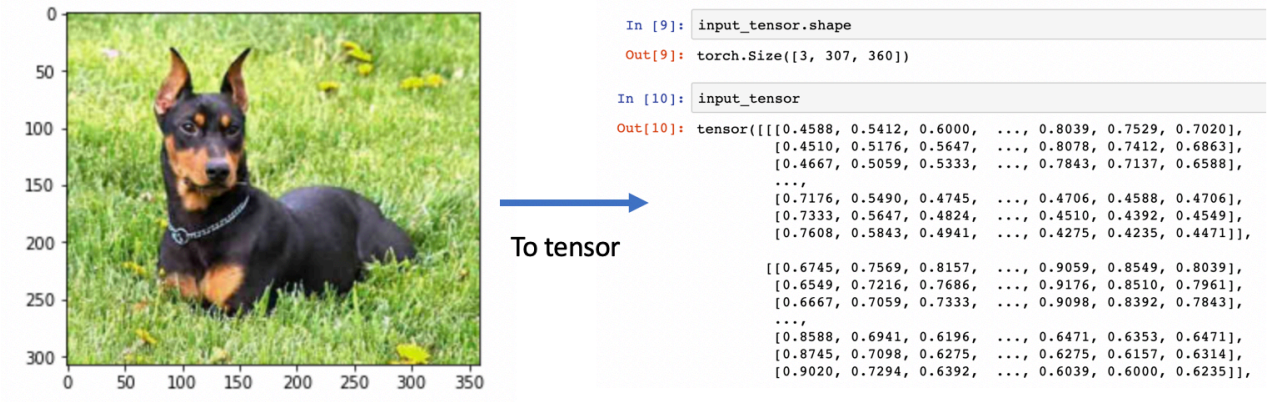
The dataset we use is *dog dataset and human dataset* that udacity provided on Github notebook[2]. We need human dataset to train our model how to detect human and also need dog dataset that our model can learn dog breeds recognition knowledges.

All datasets we use is image type, a specical type of data that can be understood as standard inputs like features as well. Firstly the original image is RGB image that each pixel is determined by three parameters(red, green and blue). The following cnn model can use these 3D features as input directly.

The dataset we use is *dogImages* and it has been splited to train, validation and test set (train:6680, valid:835, test:836 images). All the images has its corresponding breed label and images of same labal are under same folder. The are 133 classes of dog breeds, and the min, max count of training set for each class is 26 and 58. Finally, each image's size is different, so we need to pad the original image, or only cut the object that detector recognized, to make sure the input of CNN model is in same size. This step will be further explained in *Data Preprocess* part.

2.2 Exploratory Visualization

The picture below is an example of the data we use. The size of the image is 307 * 360 pixel. Each pixel is a 3D array that determine the RGB value and the color it present. If we use Pytorch's transforms to convert the input image, we will get a 3 * 307 * 360 tensor and each value is in range [0, 1]. These numerical values are the input features that we feed into the following CNN model.



2.3 Algorithms and Techniques

Based on the problem we defined, our algorithms and techniques can be devided into 2 parts, build human/dog detector and build breeds classification model.

Firstly, as for human detector, we use Haar feature-based cascade classifiers from OpenCV's open source code. This model can help us detect how many faces appear in the image and also return the location of the detected faces. So if this model return more than 1 face in the image, our face detector will return True, indicationg that this image is a human.

As for dog detector, we use a pre-trained VGG16 model. This CNN model have already trained on ImageNet dataset which contains over 10 million images and classify objects into 1000 categories. In these 1000 classes, index between 151 and 268 are labels of dog, so if the classifier predict this image's index is in this range, our dog detector will return True, indicationg that this image is a dog.

When we try to tackle image classification problem with cnn model, there are two approaches we can choose, design cnn model from scratch or use pre-trained models. The prior one can help us understand the structure of convolutional network better, but it may take too much time if we want to try out a deeper structure. So working with pre-trained models can be

more efficiency. Usually pre-trained models are more complex and usually have hundreds of layers. But most parameters are already trained and we can tune some high-level layers to make this model flexible enough for our specific problem and dataset. So in this project, I will use prior approach as benchmark model to get a sense of how cnn works, and then implements several pre-trained models to achieve a better performance on dog breeds classification.*Finally, I choose to build AlexNet model from scratch and also use pretrained Resnet50 model.*

2.4 Benchmark

In my capstone proposal, I choose relatively "simple" model, Lenet-5 and Alex as benchmark model, and the architecture of them are as the figure below. But during the coding of the project, I figured out that the input format of the LeNet-5 is not suitable for our dog dataset. Mostly, people use LeNet-5 model to play with MNIST dataset and classify hand written digits. All the input image is in grey scale which means it only has 1D value for each pixel not RGB color; and the size of the image is 28 * 28 pixel. Since our images are much larger than 28 * 28 square and all the images are in RGB color not grey scale, the simple LeNet-5 model may not be suitable for our problem. Even if we transform our image into this formatted scale to fit LeNet-5 model, we will lose much useful features for classification. So I decide to use AlexNet as Benchmark model.

Table 13-1. LeNet-5 architecture

Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully Connected	–	10	–	–	RBF
F6	Fully Connected	–	84	–	–	tanh
C5	Convolution	120	1 × 1	5 × 5	1	tanh
S4	Avg Pooling	16	5 × 5	2 × 2	2	tanh
C3	Convolution	16	10 × 10	5 × 5	1	tanh
S2	Avg Pooling	6	14 × 14	2 × 2	2	tanh
C1	Convolution	6	28 × 28	5 × 5	1	tanh
In	Input	1	32 × 32	–	–	–

Table 13-2. AlexNet architecture

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully Connected	–	1,000	–	–	–	Softmax
F9	Fully Connected	–	4,096	–	–	–	ReLU
F8	Fully Connected	–	4,096	–	–	–	ReLU
C7	Convolution	256	13 × 13	3 × 3	1	SAME	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
S4	Max Pooling	256	13 × 13	3 × 3	2	VALID	–
C3	Convolution	256	27 × 27	5 × 5	1	SAME	ReLU
S2	Max Pooling	96	27 × 27	3 × 3	2	VALID	–
C1	Convolution	96	55 × 55	11 × 11	4	SAME	ReLU
In	Input	3 (RGB)	224 × 224	–	–	–	–

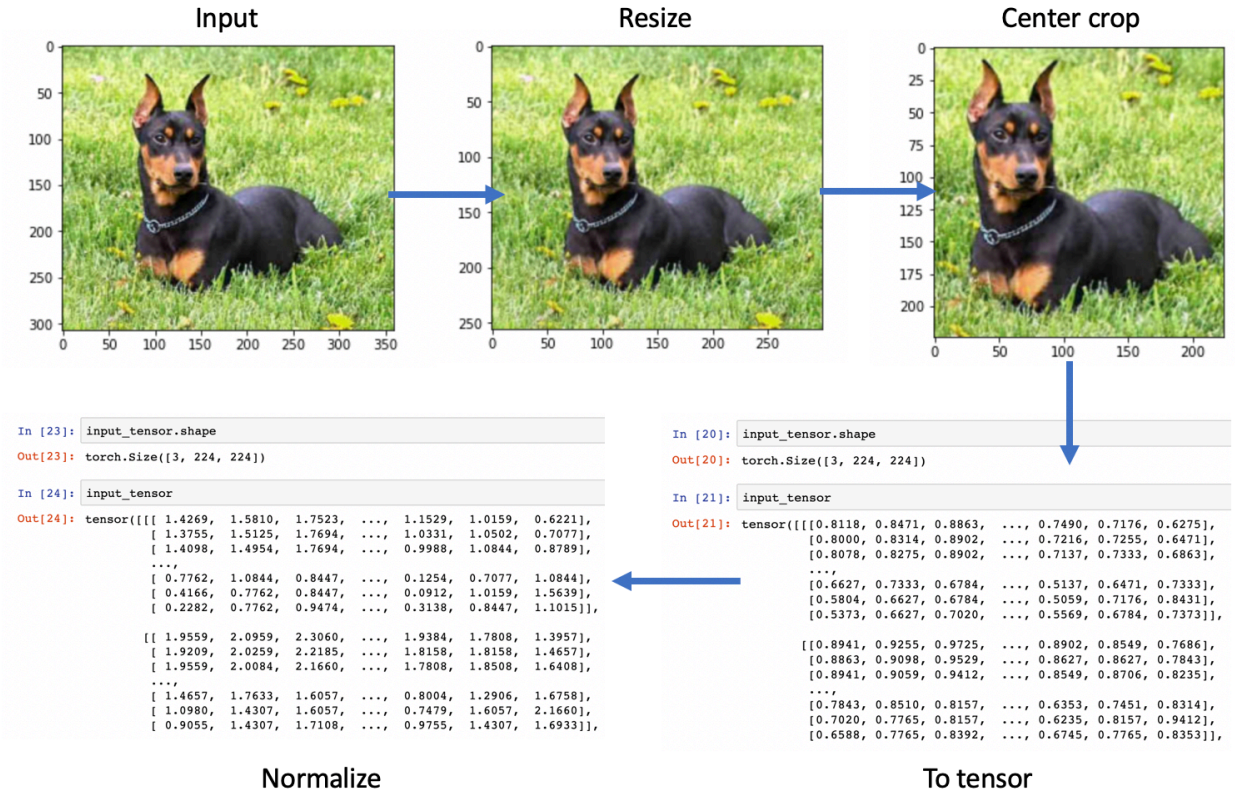
As for the famous AlexNet model, the model's first part is convolution layers and the second part is fully-connected linear layers. In the first part, we use 5 convolution layers to extract the features from images. Between each conv layers, we also ReLu as activation function and BatchNorm layer to help with faster converge during training process. Also we have MaxPool layer to extract the most significant features in the image and also reduce the image size. After conv layers, we unfold all features on 1 dimension and construct the fully-connected layers. In order to achieve our final classification task, we use these 4096 features to predict 133 classes

of breed.

3 Methodology

3.1 Data Preprocessing

My preprocess code firstly resize the image to 256 * 256 pixel because our dataset have different width and height so we need to transform them into same scale. Then I use the center crop to cut the center part of the image and the remaining size is 224 * 224. Finally we normalize the RGB value by given mean and std paramters according to pytorch documentation. Our input tensor's size is 3 * 224 * 224 because the following cnn model we use (alexnet and pretrained resnet) require this input size.



Usually for image classification problem, data augmentation is a common approach to enhance the performance of the model. Image augmentation may include randomly flips, rotations, crops... However, I didn't use data augmentation in this project because the dataset is relatively enough and training process is quiet long. And our final test accuracy is great enough so if we need to improve the accuracy further in the future, I will try to use augmentation and tune my model for more epoches.

Finally we need a Dataset class to load our dataset for following pytorch model. We use *torchvision.datasets.ImageFolder* directly since our dataset is labeled and each label is classified into a folder. We build 3 separate dataset class for train, valid, test.

3.2 Implementation

According to the cells and codes in my jupyter notebook...

- human detector

  -> load CascadeClassifier from xml files

  -> load people image and preprocess (transform to grey scale)

  -> get return results and decide if this picture is a human

  -> test the human detector performance on human and dog subset

- dog detector

  -> load pre-trained VGG16 model from torch

  -> preprocess image and get predicted classification result

  -> get return results and decide if this picture is a dog (the valid label range is 151-268)

  -> test the dog detector performance on human and dog subset

- dog breed classifier

  -> define dataset and dataloader for train, valid, train

  -> define Net() class for our scratch model architecture (for me I use AlexNet's structure)

  -> define train process: specify loss function (CrossEntrophy for this model) and optimizer (SGD optimizer).

  -> launch training job to sagemaker session: the notebook instance I use do not have enough memory for this problem, and also lack gpu resources to make training process faster. So I use sagemaker session to train this model on a $ml.p2.xlarge$ instance. We need to add a model_ scratch_ alexnet.py file as the entry point of the training job. This py file contains several important part:

  Net(), the model architecture we defined.

  _get_train_data_loader(batch_size, training_dir), get the training dataset we defined before (in data preprocessing step)

_get_test_data_loader(test_batch_size, training_dir), get the valid dataset (the *test* here means test during training process so we use our validation datasset)

train(), launch the training process for the given number of epoches, and log the loss value and accuracy for each epoch.

test(), test the model perforamce on validation set when each epoch ends.

save_model(), save the model.pth file to s3 bucket.

-> fit the sagemaker PyTorch estimator and download the model.pth file when the training job is done.

-> load the trained model parameters and test the accuracy on (real) test set.

3.3 Refinement

The implementation above we only discuss the scratch model architecture. In order to improve the model's classification performance and improve the efficiency of the training jobs, in this section we use a pre-trained Resnet50 model and the detailed implementation is as below:

-> define dataset and dataloader for train, valid, train

-> load pre-trained resnet50 model and adjust the final fully-connected layer's dimension to (2048, 133) as we need, and set other parameters' requires_grad as False, which means only the fc layer will be tuned during the training process.

-> define train process: specify loss function (CrossEntrophy for this model) and optimizer (SGD optimizer).

-> launch training job to sagemaker session: similar to steps in benchmark model but load model differently.

4 Results

4.1 Model Evaluation and Validation

- human detector performance: we test a subset of human and dog, each contains 100 images. In human subset 98% detected in human files. In dog subset 7% detected in dog files.
- dog detector performance: we test the dog detector on same subset. In human subset 1% detected in human files; In dog subset 96% detected in dog files. Both of our detector perform well.

- dog classifier performance:

  For Benchmark model, after 25 epoches of training, the training loss is reduced from 4.88 to 2.93; the validation accuracy is increased from 1% to 9%.

```
Train Epoch: 1 [6400/6680 (95%)] Loss: 4.886452
Test set: Average loss: 0.0029, Accuracy: 6/835 (1%)

Train Epoch: 2 [6400/6680 (95%)] Loss: 4.910110
Test set: Average loss: 0.0029, Accuracy: 9/835 (1%)

Train Epoch: 3 [6400/6680 (95%)] Loss: 4.802454
Test set: Average loss: 0.0029, Accuracy: 9/835 (1%)

Train Epoch: 4 [6400/6680 (95%)] Loss: 4.756186
Test set: Average loss: 0.0029, Accuracy: 14/835 (2%)

Train Epoch: 5 [6400/6680 (95%)] Loss: 4.805058
Test set: Average loss: 0.0028, Accuracy: 21/835 (3%)

Train Epoch: 6 [6400/6680 (95%)] Loss: 4.843408
Test set: Average loss: 0.0028, Accuracy: 24/835 (3%)

Train Epoch: 7 [6400/6680 (95%)] Loss: 4.629992
Test set: Average loss: 0.0027, Accuracy: 32/835 (4%)

Train Epoch: 8 [6400/6680 (95%)] Loss: 4.215341
Test set: Average loss: 0.0027, Accuracy: 28/835 (3%)

Train Epoch: 9 [6400/6680 (95%)] Loss: 4.273640
Test set: Average loss: 0.0027, Accuracy: 40/835 (5%)

Train Epoch: 10 [6400/6680 (95%)] Loss: 4.222699
Test set: Average loss: 0.0026, Accuracy: 40/835 (5%)

Train Epoch: 11 [6400/6680 (95%)] Loss: 4.040222
Test set: Average loss: 0.0026, Accuracy: 50/835 (6%)

Train Epoch: 12 [6400/6680 (95%)] Loss: 3.945643
Test set: Average loss: 0.0025, Accuracy: 46/835 (6%)

Train Epoch: 13 [6400/6680 (95%)] Loss: 3.889547
Test set: Average loss: 0.0025, Accuracy: 64/835 (8%)

Train Epoch: 14 [6400/6680 (95%)] Loss: 3.876645
Test set: Average loss: 0.0026, Accuracy: 50/835 (6%)

Train Epoch: 15 [6400/6680 (95%)] Loss: 3.688797
Test set: Average loss: 0.0028, Accuracy: 37/835 (4%)

Train Epoch: 16 [6400/6680 (95%)] Loss: 3.741326
Test set: Average loss: 0.0024, Accuracy: 66/835 (8%)

Train Epoch: 17 [6400/6680 (95%)] Loss: 3.896874
Test set: Average loss: 0.0030, Accuracy: 38/835 (5%)

Train Epoch: 18 [6400/6680 (95%)] Loss: 3.662129
Test set: Average loss: 0.0025, Accuracy: 67/835 (8%)

Train Epoch: 19 [6400/6680 (95%)] Loss: 3.405469
Test set: Average loss: 0.0024, Accuracy: 71/835 (9%)

Train Epoch: 20 [6400/6680 (95%)] Loss: 3.279269
Test set: Average loss: 0.0025, Accuracy: 72/835 (9%)

Train Epoch: 21 [6400/6680 (95%)] Loss: 3.284459
Test set: Average loss: 0.0025, Accuracy: 67/835 (8%)

Train Epoch: 22 [6400/6680 (95%)] Loss: 3.299536
Test set: Average loss: 0.0024, Accuracy: 89/835 (11%)

Train Epoch: 23 [6400/6680 (95%)] Loss: 2.997395
Test set: Average loss: 0.0024, Accuracy: 93/835 (11%)

Train Epoch: 24 [6400/6680 (95%)] Loss: 3.107534
Test set: Average loss: 0.0024, Accuracy: 84/835 (10%)

Train Epoch: 25 [6400/6680 (95%)] Loss: 2.930168
Test set: Average loss: 0.0025, Accuracy: 73/835 (9%)

Saving the model.
```

  If we test our benchmark model on test set, we got the test accuracy of 10.885167% (91/836). This is greater than 10% as requirement, but it is not good enough for our app. So we test the pre-trained model in the following section.

4.2 Justification

For pre-trained model, after only 20 epoches of training, the training loss is reduced from 3.35 to 0.43; the validation accuracy is increased from 57% to 87%. If we test our pre-trained model on test set, we got the test accuracy of 88.038278% (736/836). So the pre-trained model has much higher accuracy and is much more robust than the benchmark showed above. We will use the transfer model in the final app algorithm.

```
Train Epoch: 1 [6400/6680 (95%)] Loss: 3.353631
Test set: Average loss: 0.0019, Accuracy: 472/835 (57%)

Train Epoch: 2 [6400/6680 (95%)] Loss: 2.227427
Test set: Average loss: 0.0013, Accuracy: 578/835 (69%)

Train Epoch: 3 [6400/6680 (95%)] Loss: 1.678176
Test set: Average loss: 0.0009, Accuracy: 625/835 (75%)

Train Epoch: 4 [6400/6680 (95%)] Loss: 1.427252
Test set: Average loss: 0.0008, Accuracy: 661/835 (79%)

Train Epoch: 5 [6400/6680 (95%)] Loss: 0.986263
Test set: Average loss: 0.0006, Accuracy: 683/835 (82%)

Train Epoch: 6 [6400/6680 (95%)] Loss: 0.773723
Test set: Average loss: 0.0006, Accuracy: 699/835 (84%)

Train Epoch: 7 [6400/6680 (95%)] Loss: 0.902008
Test set: Average loss: 0.0005, Accuracy: 701/835 (84%)

Train Epoch: 8 [6400/6680 (95%)] Loss: 0.704648
Test set: Average loss: 0.0005, Accuracy: 711/835 (85%)

Train Epoch: 9 [6400/6680 (95%)] Loss: 0.700431
Test set: Average loss: 0.0004, Accuracy: 715/835 (86%)

Train Epoch: 10 [6400/6680 (95%)] Loss: 0.690981
Test set: Average loss: 0.0004, Accuracy: 710/835 (85%)
```

```
Train Epoch: 11 [6400/6680 (95%)] Loss: 0.725048
Test set: Average loss: 0.0004, Accuracy: 712/835 (85%)

Train Epoch: 12 [6400/6680 (95%)] Loss: 0.648235
Test set: Average loss: 0.0004, Accuracy: 727/835 (87%)

Train Epoch: 13 [6400/6680 (95%)] Loss: 0.441636
Test set: Average loss: 0.0004, Accuracy: 725/835 (87%)

Train Epoch: 14 [6400/6680 (95%)] Loss: 0.536662
Test set: Average loss: 0.0004, Accuracy: 725/835 (87%)

Train Epoch: 15 [6400/6680 (95%)] Loss: 0.358337
Test set: Average loss: 0.0004, Accuracy: 725/835 (87%)

Train Epoch: 16 [6400/6680 (95%)] Loss: 0.497000
Test set: Average loss: 0.0003, Accuracy: 729/835 (87%)

Train Epoch: 17 [6400/6680 (95%)] Loss: 0.432177
Test set: Average loss: 0.0003, Accuracy: 719/835 (86%)

Train Epoch: 18 [6400/6680 (95%)] Loss: 0.428928
Test set: Average loss: 0.0003, Accuracy: 722/835 (86%)

Train Epoch: 19 [6400/6680 (95%)] Loss: 0.373278
Test set: Average loss: 0.0003, Accuracy: 730/835 (87%)

Train Epoch: 20 [6400/6680 (95%)] Loss: 0.432828
```

5 Conclusion

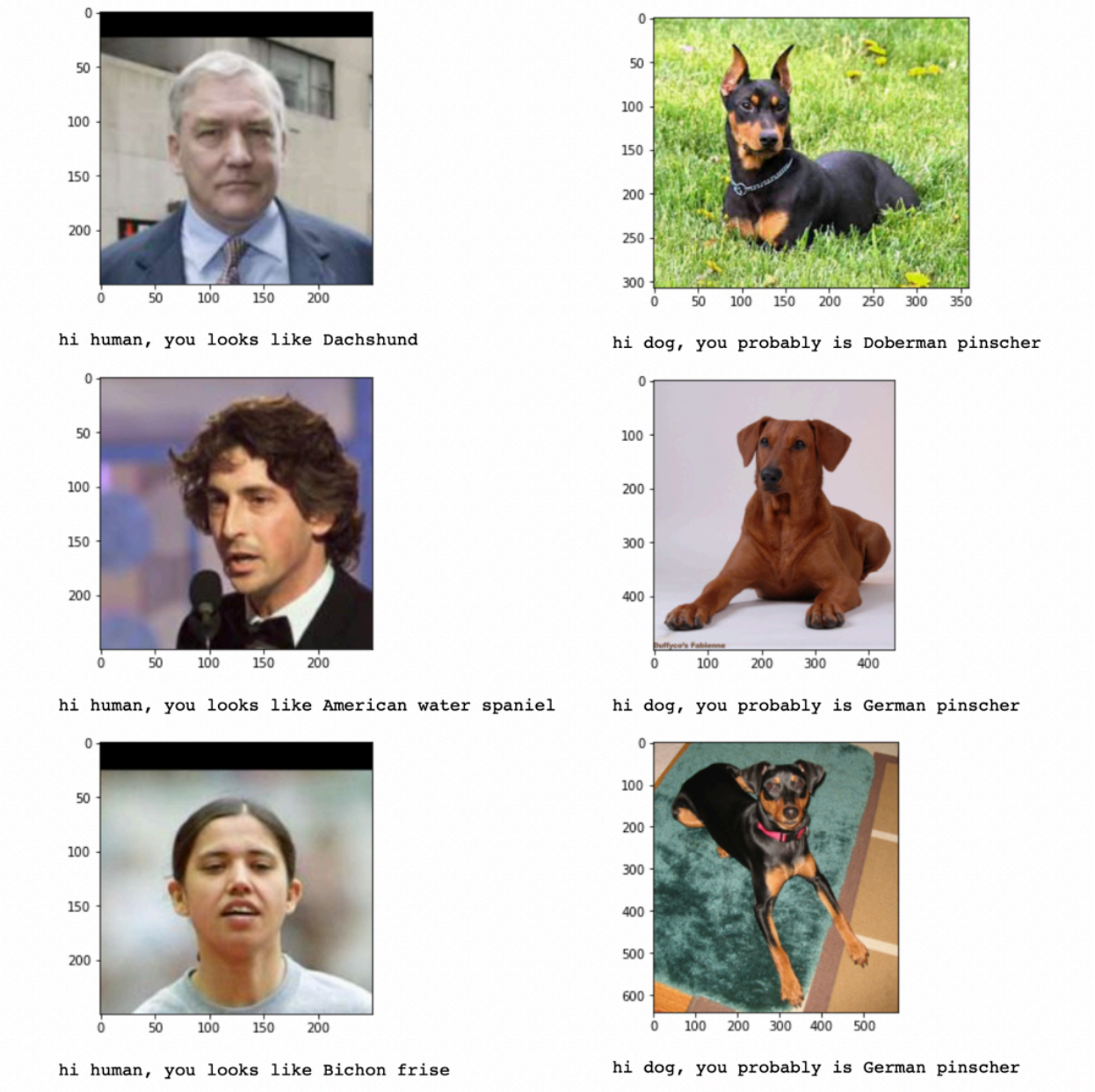
5.1 Free-Form Visualization

As the figure shows, our app can 1) detect dog and give the possible breed of the dog. 2) detect human and give the similar dog breed.

5.2 Reflection

To sum up, in this project I:

- Use OpenCV's model and pre-trained VGG16 model to build a human/dog detector.
- Use Pytorch framework to build a whole process of training and use a deep learning model. From data preprosses, define model architecture and define train/test process to achieve final results.
- Learned how to build CNN model from scratch and also tuned pre-trained model. By compare my benchmark model and pre-trained model, I figured out that mostly use pre-trained model may achieve better performance with higher efficiency.
- The most difficult part of this project is when I tried to train CNN model in current notebook instance, I always got memory error and do not have gpu to use since the limitation of the instance type. So I have to add a train script and use sagemaker session to launch training jobs to the suitable instance. This process is the most difficult part



over all my project, but thanks to the previous project in ML nanodegree, I fixed all the bugs and successfully finished all the training jobs.

5.3 Improvement

There are still some improvement of my algorithm:

- There are some similar breeds that people may find hard to recognize too. So we can use data augmentation or more complex cnn model to improve our accuracy on these similar breeds.
- For some images, the dog or human may not appear at the center of the picture. So we can firstly build a better object detector and then corp the image based on the object

accurate location and use this part image as the classifier's input.

- We can deploy our model by sagemaker endpoints or lambda so other users can upload their images and try the classifier too.

References

[1] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow, Chapter 13*. O'Reilly Media, Inc.

[2] `https://github.com/udacity/deep-learning-v2-pytorch/tree/master/project-dog-classific` Udacity.