Machine Learning Engineer Nanodegree

Capstone Project

Yitao Hu
August 11th, 2019

CNN Project: Dog Breed Classifier

I. Definition

Project Overview

Correctly classifying objects captured by vision is a basic but crucial function for human brains, because we can only perform corresponded “right” actions if we can precisely identify the object. For the same reason, image recognition also plays a crucial role in the state-of-art artificial intelligence projects or machine learning pipelines. Only when the autopilot of a self-driving car can effectively identify various objects including pedestrians, traffic lights, and other vehicles, it could correctly stay or change the lanes, accelerate or brake, and make a turn.

However, one issue for both human brains and machine learning algorithms is to identify objects that share great similar intra-class patterns. Identity recognition is one case. For instance, because eastern Asians share very similar biological traits such as black hair and eyes, a body size smaller than the white, and relatively flat facial shape, most westerns find it difficult to identify an eastern Asian or even correctly classify one as a Chinese, Japanese or Korean. The same is also true vice versa. Therefore, although this project aims to build an algorithm that only focuses on solving one particular intra-class classification problem, identifying dog breed, the same hypothesized model, learning algorithm, and even learned parameters can be generalized to other fine-grained image classification projects.

In this project, with the template Jupyter notebook and image data provided by

Udacity team, I tried to build dog breed classification algorithm based on a convolutional neural network or CNN model, a particular type of neural network whose design was developed to mimic that of the human visual cortex. As a CNN utilizes filters, like visual cortex in our brains, to extract high-level features such as shapes of various parts, and max-pooling layers to perform dimension reduction, "(it) leads to savings in memory requirements …(and) even outperform humans in cases such as classifying objects into fine-grained categories such as…breed of dog or species of bird", according to a paper from IP Group, Cadence. In other words, because a CNN simulates visual mechanism in human brains, it enjoys similar advantages, including memory, saving and feature extracting, and thus is expected to perform as well as, if not better than humans in classification tasks. In the same paper, the researchers also mentioned several cases, in which a CNN classifier achieved the highest correct detection rates, including a 99.77% in MNIST handwriting recognition, a 97.47% with the NORB dataset.

Problem Statement

The problem to be solved in this project is building a dog breed classification algorithm with higher than 60% accuracy to be implemented in a web or smartphone app. In particular, this algorithm takes an image as an input and returns an estimated dog breed if a dog is detected in the image, or returns a dog breed the human look like most if a human face is detected.

The intended solution to this problem consists of two classifiers. The first classifier detects the existence of a human face or a dog in the image, and then the second classifier tries to figure out the corresponded dog breed or the dog breed the person most resembles.

Following guidelines offered by Udacity team, I attempted to tackle this issue by using transfer learning techniques with Pytorch pre-trained models, so that I can use utilized the parameters and architecture already trained by millions of images, and only need to fit the few hundreds of parameters based on my relatively small image dataset.

Metrics

The evaluation metrics are the test multi-class Cross-Entropy Loss, which is also the optimization object, and test accuracy. The multi-class Cross-Entropy Loss is defined by the formula from Pytorch organization: Loss (x, class) =-x[class]+log(j ∑exp(x[j])). For the test accuracy, it is the percentage of correctly classified images in the test set.

In the data exploration process, I discovered that the dog images are not skewed to one or several dog breeds. As a result, I decided to evaluate the model performance based on simple and intuitive accuracy instead of complicated metrics such as F1 score. As for the Cross-Entropy Loss, it not only covert binary classification measurement to continuous numerical values, which is more convenient for optimization, but also provides an intuitive interpretation in terms of probabilities. Therefore, I chose it as another evaluation metrics and the optimization object.

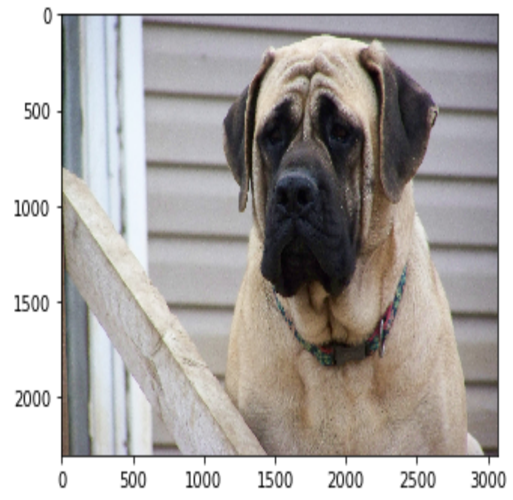
II. Analysis

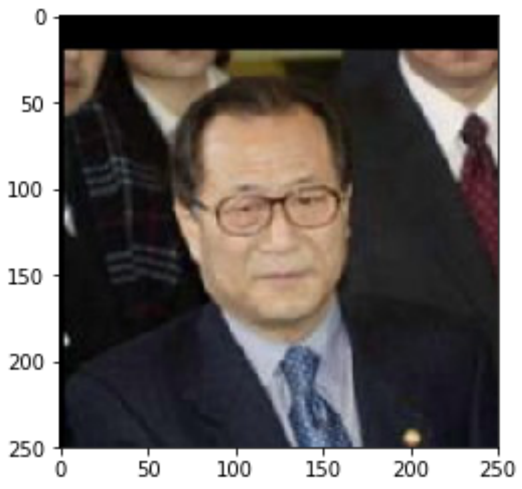
Datasets and Inputs Source

The project is developed completely from the ipython Jupyter Notebook "dog_app", provided by Udacity team, which contains some template codes, guidelines, and suggestions on implementation. Besides, I added some additional code and markdown cells for other steps such as data exploration and pre-processing. All the image data are provided by Udacity team, including 13,233 human faces and 8,351 labeled dog RBG jpg images of 133 breeds. Besides these data, I used 6 pictures of my portraits and took in my life for testing the final algorithm. The dog images are divided into 6,680 in training set, 835 in the validation set, and 836 in the test set, while the human images are purely used to test the human face detector.

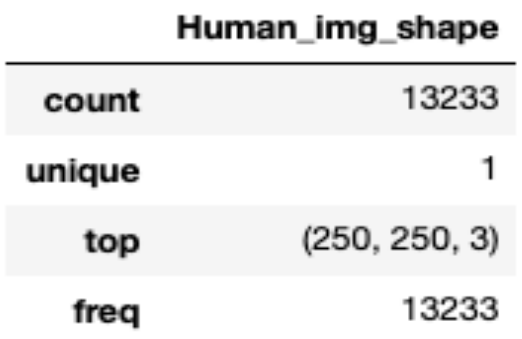
Data Exploration

By displaying some samples from dog and human face images, I discovered several interesting properties. First, both dog and human face images are of portrait type, which means only one figure or face exists in the image to be classified. This implies that I do not need to drop any images of multiple figures, which will greatly confuse the classification algorithm. A sample image of a dog and a human face is shown below:

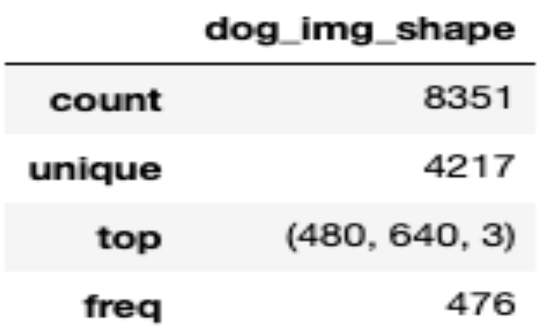




Second, by further extracting and displaying the shape of each image, I discovered that the dog images are RBG jpg with size ranging from hundreds by hundreds to thousands by thousands of pixels, whereas human images are uniformly 250 by 250. This indicates that the dog images have to be pre-processed, including resizing and augmentations into a standard format before used in training, validation, and testing. The following is some descriptive statistics about the size of the dog and human face image shapes:



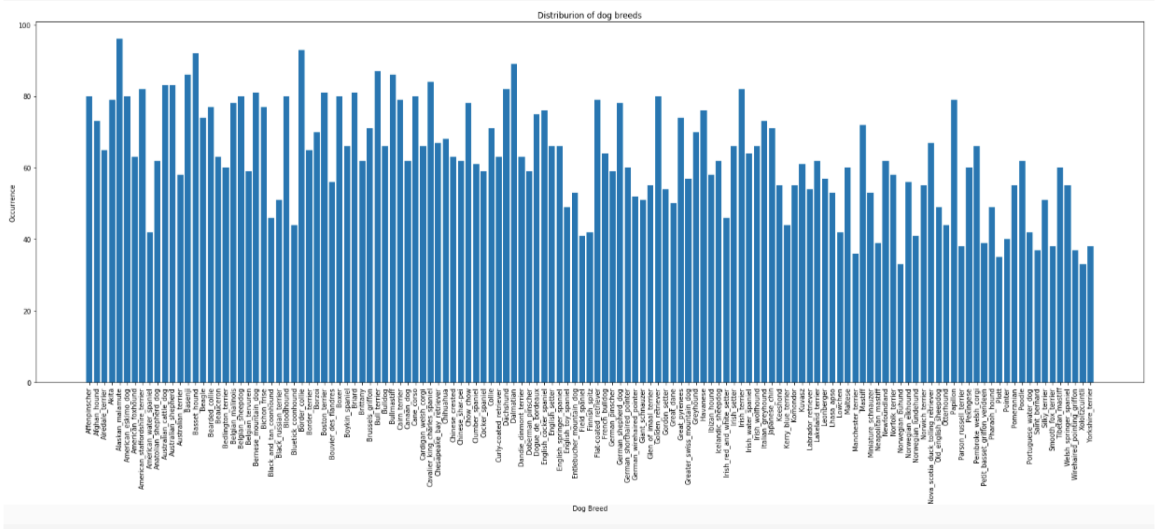
	Human_img_shape
count	13233
unique	1
top	(250, 250, 3)
freq	13233



	dog_img_shape
count	8351
unique	4217
top	(480, 640, 3)
freq	476

Exploratory Visualization

By writing a function that extracts dog breeds from the image directories and plotting the number of occurrence over various dog breeds on a bar chart, I found that the dog images are fairly uniformly distributed, and so I regarded accuracy as a fair and intuitive evaluation metrics on model performance because no issues of skewed data need to be handled. The distribution of dog breed bar chart is shown below:



Algorithms and Techniques

As mentioned above, the final solution consists of two classification algorithms: a human face and dog detector, and a dog breed detector. To be more specific, the first detector is set to be based on a pre-trained OpenCV's and Pytorch VGG16 CNN model provided by Udacity team for this project.

The reason for using pre-trained model is to exploit the parameter and hyper-parameters learned from millions of images where these models come from. In particular, the VGG16 Pytorch model was learned from ImageNet, an image dataset of over 10 million images from more than 1000 categories. Therefore, compared with building a classification from scratch based on thousands of examples, employing a pre-trained model tends to have a much better performance. Besides, although the inputs image data are not used to train the first detector, they are actually utilized to test the performance of the pre-trained models.

Almost for the same reason, following instructions of the Udacity team, I employed transfer learning techniques to build the model for the second detector. However, there are a few technical differences in this case. Although here I also exploited the net architecture and most parameters built on ImageNet image dataset, I substituted the original output layer with a fully connected linear transformation layer with 133 output nodes, consistent with the number of classes I intend to classify examples in. This process is called fine-tuning in transfer learning. In addition, all labeled dog images in the training set were used here to fit the 133 parameters in the rebuilt final layer. As for the validation and test set, they were used to evaluate the performance of a number of different transfer learned models, and then to select the best performer. The merit of this arrangement or technique is that because I can use part of the pre-trained model to extract all low-level features such as shapes, which are very similar in image recognition, all the labeled dog images data can be effectively exploited to fit the final 133 parameters. In other words, instead of fitting millions of parameters in training a complex CNN from scratch, I now only need to fit 133 parameters with all the 6,680 training dog images. As a result, this technique not only tackles data sufficiency problem and increases test accuracy, but also greatly reduces computational workload.

Benchmark

The benchmark in this project is test performance of a simple-structure CNN model I developed from scratch and trained after 30 epochs. This simple CNN model was developed from a sample architecture provided by Udacity team, which contains 3 convolutional layers and one linear layer, including a total number of 19,189 trainable parameters. When training this model, I observed a very severe over-fitting phenomenon, where training error decreased constantly to convergence, while validation error rebounded after the first few epochs. As a result, I included a batch norm layer after each convolutional layer as a regularization technique, and an additional linear layer and dropout layers to increase classification accuracy. The modified simple CNN model finally yields a test accuracy of 19 % and Cross-Entropy Loss of 3.73

I understand that this performance is far from acceptable mainly because of the simple architecture and limited training examples, but I would still argue that it plays a decent role as a benchmark. In this particular case, this unimpressive performance demonstrates how limited performance we can achieve by fitting a larger number of parameters on much smaller data size. As a result, when compared with the performance of transfer learned models, it also shows how much we can improve by tackling data sufficiency issue through transfer learning techniques.

III. Methodology

Data Pre-processing

To address the various dog images size issue and to convert images to the required tensor data type, I performed the data pre-process steps when building the data loaders for both the pre-trained VGG16 and transfer learned model. To be more specific, I first used torchvision's ImageFolder function to read in the images from image directories. Then, by defining a transforms object in torchvison library, I resized and crop each image to a standard shape of 224 by 224 by 3, the minimal shape that can be used by Pytorch pre-trained CNN models, and then converted each image to a normalized numerical tensor matrix. This process corrected the issue of various dog image shape and convert the image into a model readable format.

For the training set only, I also augmented the images by randomly flipping and rotation. This is basically another trick to avoid over-fitting by adding more random noise in the training set. Also, to make the benchmark and solution model performance comparable, I used the same data loader or pre-processing procedure.

Implementation

To build the first human face and dog detector, I first imported the pre-trained models from Pytorch libraries and directories specified by Udacity team. Afterward, I tested these models on 100 images, and, after obtaining satisfactory accuracy, I assembled these models into two functions that return Boolean values when a human face or a dog is detected. These functions can be used to build a final pipeline for the final algorithm.

For both the benchmark and solution model, I defined identical loss function, optimizer, training and test function to make their performance comparable. To be more specific, I defined the net layers for the benchmark from scratch after consulting sample model structure, whereas the solution architecture was borrowed from several Pytorch pre-trained models.

The training function takes 30 times epochs, 6,680 training examples, Cross-Entropy Loss function and Stochastic Gradient Descent optimizer as inputs, and then iterates the process of performing forward and backward prorogation, calculating loss terms, updating parameters and saving the model artifacts to local directories whenever validation loss decreases for 30 times. After training, the testing function takes the saved model artifacts and test data makes an inference and then prints out the test accuracy and Cross-Entropy Loss.

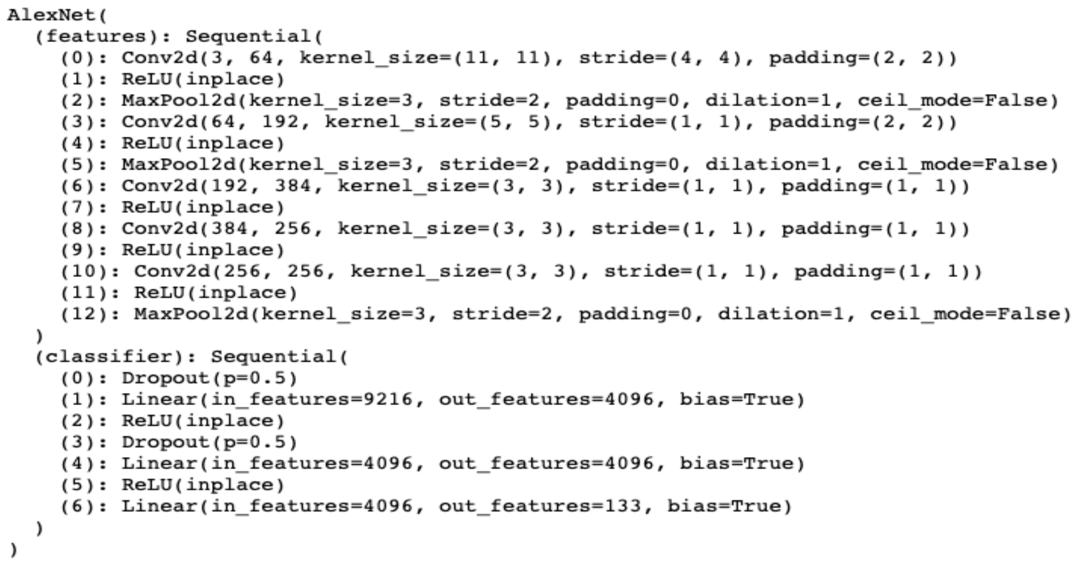
After completing a satisfactory model for classifying dogs into the corresponded breeds, the model artifact is deployed into a third function that takes an image and returns a string value of dog breed. In the last step, all three functions are assembled into a single function called “run_app”, which is the final algorithm.

One issue I found when training the transfer learned model is the divergence of training and validation loss. Although both the training and the validation loss consistently decreased during the 30 epochs’ iteration, training loss went down at a much higher speech than validation loss.

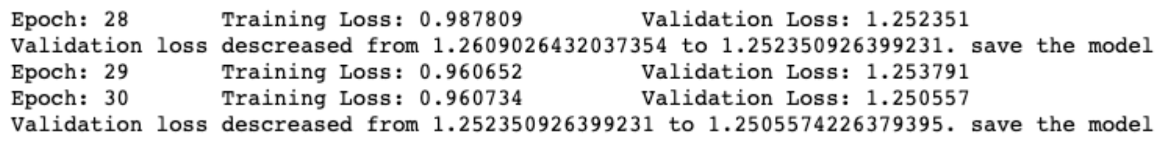
Refinement

The improvement is a process of iterating different combinations of hyperparameters and pre-trained Pytorch models and then choosing the best performer.

My initial solution was to build a transfer learned model from a pre-trained AlexNet, which has a relatively simple architecture, with a learning rate of 0.01. The aim to reduce computational workload and make the classification converge quickly, as my total number of iterations is only around 30. The architecture and last three epoch loss stats of this solution are provided as follows:



```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Dropout(p=0.5)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace)
    (3): Dropout(p=0.5)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace)
    (6): Linear(in_features=4096, out_features=133, bias=True)
  )
)
```



```
Epoch: 28       Training Loss: 0.987809         Validation Loss: 1.252351
Validation loss descreased from 1.2609026432037354 to 1.252350926399231. save the model
Epoch: 29       Training Loss: 0.960652         Validation Loss: 1.253791
Epoch: 30       Training Loss: 0.960734         Validation Loss: 1.250557
Validation loss descreased from 1.252350926399231 to 1.2505574226379395. save the model
```

However, as mentioned above, in spite of a signal of convergence for both training and validation loss, training loss decreased much more quickly than the validation loss, which shows an inclination of over-fitting.

To alleviate this issue, I used a much smaller learning rate of 0.001 to train the second candidate model. Besides, a pre-trained DenseNet 121 was employed this time. Although to avoid over-fitting, it is seemingly inappropriate to use such a complex net on a relatively small dataset, it is not the case in transfer learning. This is because regardless of the architecture, all the training data are utilized only to fit the 133 parameters in the last linear layer. In contrast, a net with a larger number of layers may be more advantageous, since more feature-extracting parameters can be directly transplanted from a pre-trained model, and thus contribute to the final performance.

The validation performance confirmed my reasoning. With a transferred DenseNet 121 architecture, the training and validation loss declined relatively as the same speed, and the trained model yields a much lower validation loss of 1.12 after 30 epochs.

To find a further improved solution, I tried an even more complex ReseNet 152 architecture and got an even better performance, which I decided to use as my final solution.

IV. Results

Model Evaluation and Validation

As discussed above, the final transferred ResNet 152 model was developed during the processing of trying transferring various pre-trained CNNs and picking the best performer in terms of validation loss. In particular, the transferred ResNet 152 was obtained after 30 epochs with a Cross-Entropy Loss of 1.01 for validation. It was chosen also because of its merit of fast convergence and impressive performance with an error rate of 4.49 % on ImageNet validation set according to a post on the Medium written by Prakash Jay. On the test set, it achieved a test loss of 1.03 and an accuracy of 81%, which I would consider satisfactory because classifying dog breed is an intra-class fine-grain task.

To further test the robustness of the final model, I tried to randomly shuffle the training, validation, and test set for several times and obtained similar test performance. Also, because there is no chronical or sequential relationship between training, validation and test data, and test performance was not used to choose the model architecture, any data leakage is unlikely. As a result, I would argue the test performance of the final model can be generalized.

However, it should be mentioned that because the data used in this project are portrait type images, the algorithm will tend to malfunction if app users pass in an image with multiple figures.

Justification

Compared with the simple CNN benchmark model, which only achieved a test loss of 3.73 and accuracy of 19%, the final transfer learned model significantly improved the performance with a test loss of 1.03 and an accuracy of 81%.

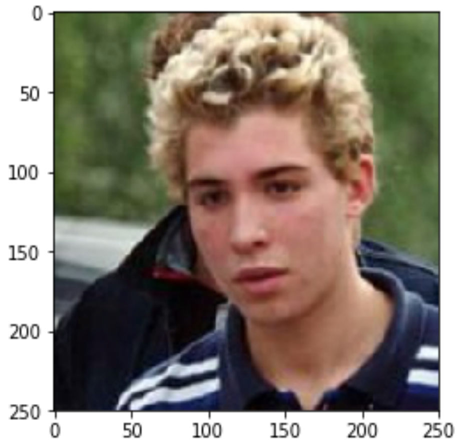
Theoretically speaking, transfer learning is especially advantageous when the pre-trained model was obtained from a much larger dataset with similar characteristics, as the new model can utilize most parameters learned from a much larger dataset it cannot use for training directly. This is exactly what happens in this case where I only have thousands of images for training whereas ResNet 152 was learned from millions of images from different objects. Because the new model can exploit low-level features already extracted by the ResNet 152 model and focus on training the final 133 parameters, it achieved a much better performance than that of the benchmark, which trained all parameters from a small dataset.

Although an 81% accuracy is far from perfection, I would consider this solution acceptable to solve the dog breed classification problem, since even a human may also have difficulties in distinguishing objects that look alike but from different sub-classes, and having such a high classification accuracy.

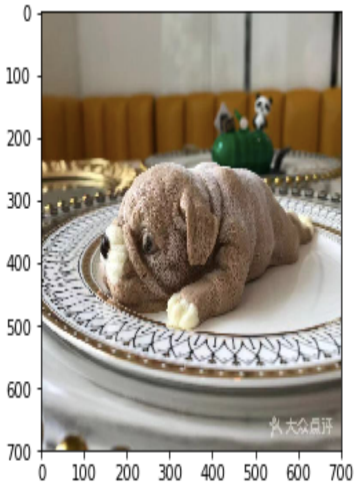
V. Conclusion

Free-Form Visualization

The following is the results I obtained after testing the final algorithm with some sample images.



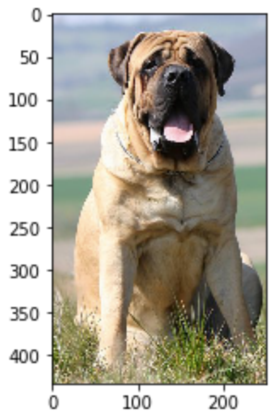
```
Hello, human!
You look like a...
Cane corso
```



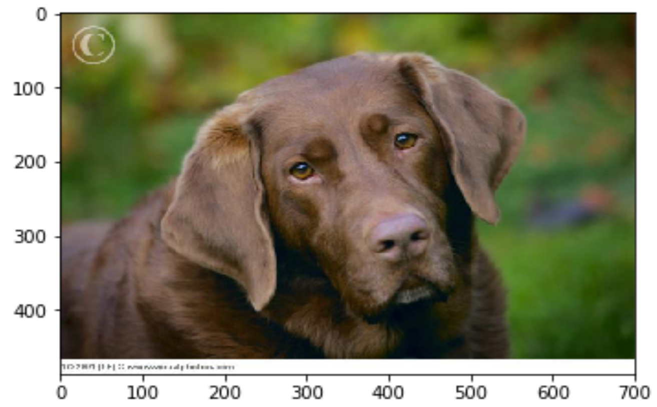
```
Error, the image is neither a human nor a dog!
```

In particular, the algorithm did well in detecting dogs and non-dog objects. During the testing period, I surprisingly discovered that it can even distinguish a real dog from a very resembling dog-shaped cake, which is far above my expectation.

However, I also observed the algorithm has the issue of misclassifying dogs into an incorrect breed with similar-looking because it cannot detect subtle differences. For instance, the algorithm misclassified test image of a normal Mastiff into a Bullmastiff, which shares a quite similar outlook. Besides, it also has a low tolerance for intra-class variations. To be more specific, although the classifier can correctly identify typical a black or yellow Labrador, it has difficulties in classifying a less common chocolate Labrador.



```
Hello, dog!
I think you are a...
Bullmastiff
```



```
Hello, dog!
I think you are a...
Chesapeake bay retriever
```

Reflection

My project begins with data exploration, where I found I needed to resize all images into a standard format, but did not need to drop any abnormal examples. Afterward, I built a human face and a dog detector basing on pre-trained OpenCV's and VGG16 Pytorch model provided by Udacity team. Then, I pre-processed the image data during the process of building data loaders. The next step is to build, train and test the benchmark model for the dog breed classifier, a shallow-layer CNN from scratch. Once the benchmark was done, I tried different pre-trained Pytorch models to build, train and validate a transfer learned model. Next, I picked the best validation performer as the final dog breed classifier. In the final step, I assembled all the human face detector, the dog detector, and the breed classifier into the final algorithm for the app, and tested it with several sample images from Udacity workplace and taken by myself.

Building the benchmark CNN model is especially challenging for me. The first benchmark CNN I built displayed serious over-fitting issue and obtained a test accuracy of around 1.6%, which is a blind guess. However, after I used more regularization techniques such as increasing the drop out rate, the model's training and validation loss, in return, failed to converge. Not until I tried more than ten times with different architecture and hyper-parameters after reading tons of posts, the model had yielded a test accuracy of 13% that can be used as a benchmark and makes sense.

The performance of the final algorithm is within my expectation. Although it can correctly classify a dog with an obvious unique outlook, it incurred errors when trying to classify dogs from the same breed but look differently such as a black Labrador and a cheery one. The same is also true for a human being. I think this issue can be further medicated by including additional labeled training examples so that the classifier can also detect subtle changes in dogs look alike.

Improvement

As mentioned above, one possible improvement for this project is to include additional labeled training and validation examples to train the model. If I could have collected a larger volume of labeled dog images, the dog breed classification algorithm can be considerably improved simply as the model can be taught to detect subtle differences. As a result, the model can distinguish the dog from breeds that look alike, and tolerate intra-breed differences.

Another thing I would do, if I had a better GPU, is to train the model with a greater number of epochs. When training the final model transferred from ResNet 152, even though the training and validation loss declined at a much slower speed, it consistently maintained a downward trend after 30 epochs. Hence, I expect to have a better performing classifier after additional training epochs.

For the same reason, I also think increasing the size of the input images is an option worth trying. With more pixels as training inputs, the trained classifier would be more likely to capture subtle inter-breed differences and tolerate intra-breed variations and thus has a better performance.

References

[1] Hijazi, Kumar, and Rowen. 2015. “Using Convolutional Neural Networks for Image Recognition.” IP Group, Cadence.
https://ip.cadence.com/uploads/901/TIP_WP_cnn_FINAL-pdf

[2] “CrossEntropyLoss”. torch.nn. Docs. Pytorch.
https://pytorch.org/docs/stable/nn.html#crossentropyloss

[3] Jay, Prakash. “Understanding and Implementing Architectures of ResNet and ResNeXt for state-of-the-art Image Classification: From Microsoft to Facebook [Part 1]”. Medium. https://medium.com/@14prakash/understanding-and-implementing-architectures-of-resnet-and-resnext-for-state-of-the-art-image-cf51669e1624