Udacity Machine Learning Engineer Nanodegree Program
Yueheng Zhong
06/11/2020

Capstone Project Report

Definition

Project Overview

Image Classification is one of the most classic problem in the machine learning field. “The goal of image classification is to predict the categories of the input image using its features. There are various approaches for solving this problem such as k nearest neighbor (KNN), Adaptive boost (Adaboosted), Artificial Neural Network (NN), Support Vector Machine (SVM).” (Thai, Hai, Thuy 2012)

For Udacity Nanodegree for Machine Learning Engineers, I chose to implement a dog breed classifier which takes the input of an image, and output the estimated breed of the dog/human in the image. The goal is to build a real-world pipeline that would take in user-supplied data and output desired results. In this project, Udacity has supplied us with the training data, which includes files of human faces as well as dog images. At the end of the project, I’ve also downloaded some test image online in order to test the algorithm that I’ve built.

Problem Statement

In this project, we want to build an dog identification app which takes a user-supplied image and output the dog breed. It then determines if the image contains a human, a dog, or neither. If the app decides that the image contains a human face, it will then run a model which takes in the image, and output a resembling dog breed of that human. If the app determines that the image contains a dog, it will try to determine and output the dog breed. If neither is detected in the image, it will output a statement, saying that neither a human nor a dog is detected in the image.

Based on the problem defined above, there are several steps that we need to complete, in order to solve the problem. First, we will need an algorithm to take in the image. Then we will need an algorithm to determine if a human is presented in the image. After that, we will need another algorithm to determine if a dog is presented in the image. Then we will need a dog breed classifier which can take in the image, and determine which dog breed the image contains. We will also need to build an algorithm which will take in all those steps and combine them together in order for the whole algorithm to work.

Metrics

The Metric that we use would be mostly based on the accuracy. We would evaluate the percentage of the human detected in human file or dog file to determine the accuracy of human detector. We would use similar method, measuring the performance of dog detector by looking at the percentage of dog detected in human files or dog files.

After that, we will continue to evaluate the dog breed classifier by looking at the accuracy on test data, that is the percentage of the dog correctly classified with its corresponding breed. I decided to choose this metric to evaluate the performance of the algorithm, mostly because this is an image classification problem. Therefore I want to achieve the best accuracy in this model.

Analysis

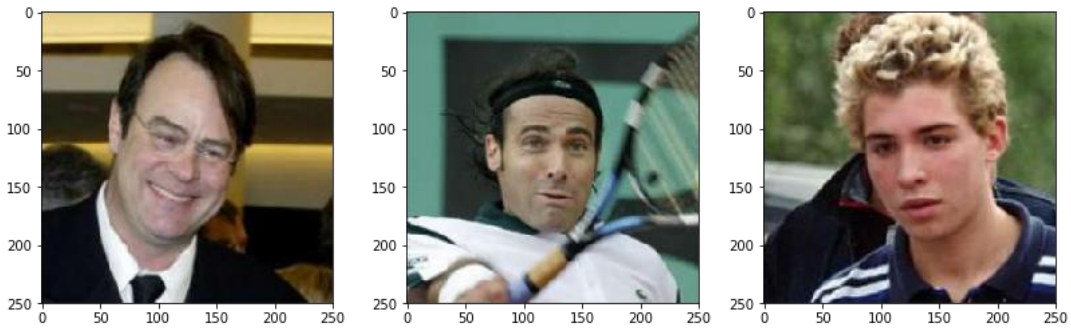
Data Exploration

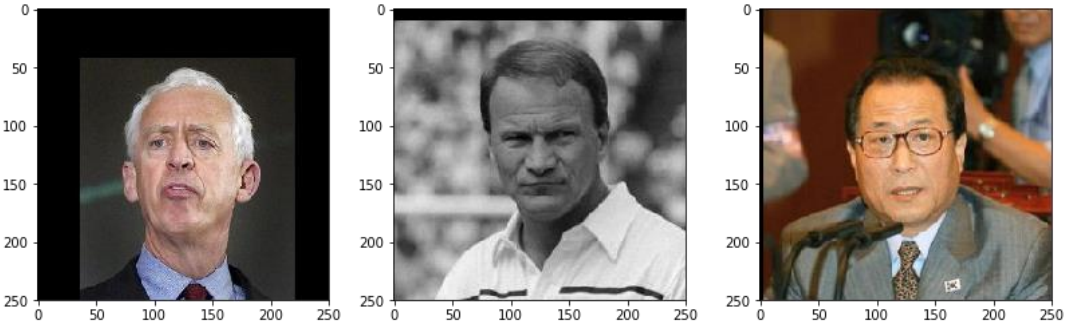
The input dataset for this project is provided by Udacity. The dataset contains both human file as well as dog file, which are input as images. The human file contains 13233 images of human faces, of a total of 5749 people. Because there can be several images of same person, the dataset for human faces is not balanced.

Dog file contains 8351 images, which is separated into 6680 images as training set, 835 images (10%) as validation set, and 836 images (10%) as test set. In the training set, it contains 133 type of dog breeds, and the size of image of different dog breed is different. Since the image size for different dog breeds ranges from 26 to 77, the training data that we have is also unbalanced.

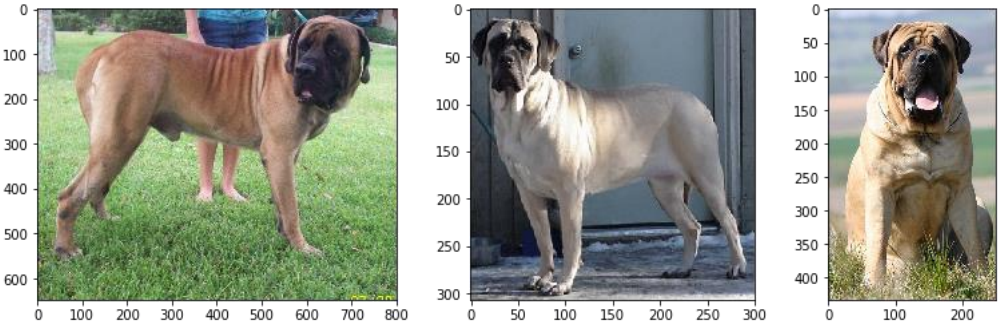
In the human files, all images are formatted in to a 250*250 RGB images. The dog file contains less standard images, each with different size, although all of them are of RGB format. The average height and width of the dog images are around 530*567, although the height can range from 113 to 4003, and width can range from 105 to 4278. It can be clearly seen that the size of the dog file is not standardized and we need to take care of that in future implementation.

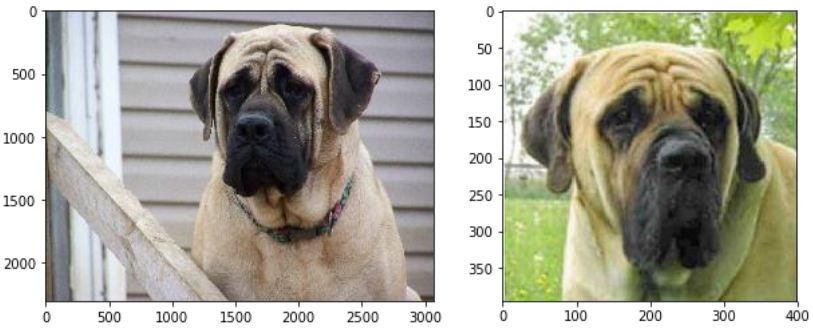
Here are some sample images for human file:





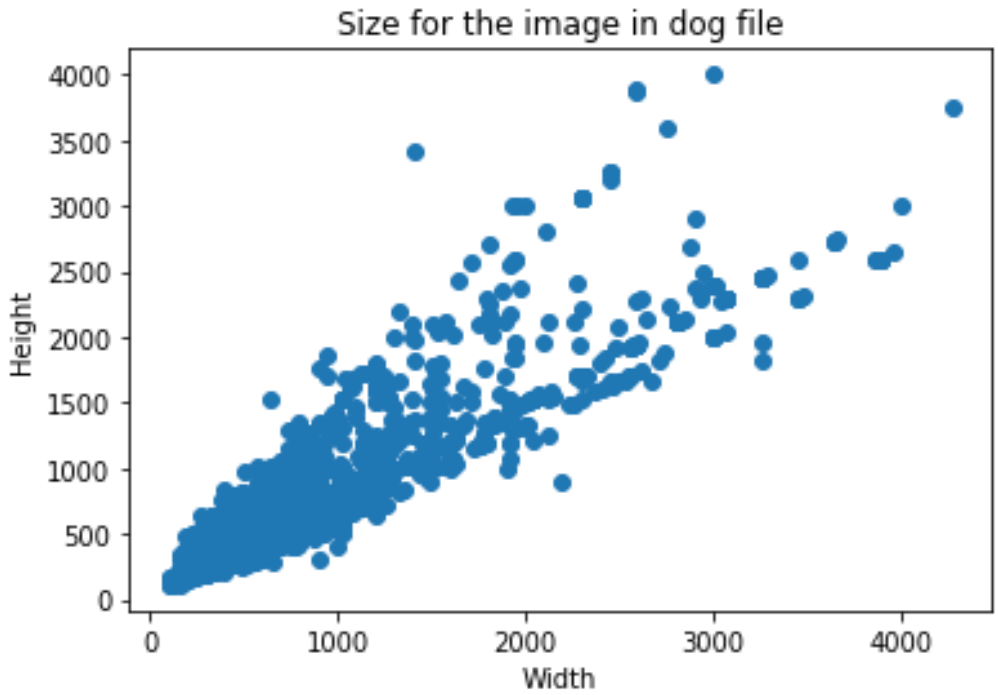
Here are some sample images in the dog file:



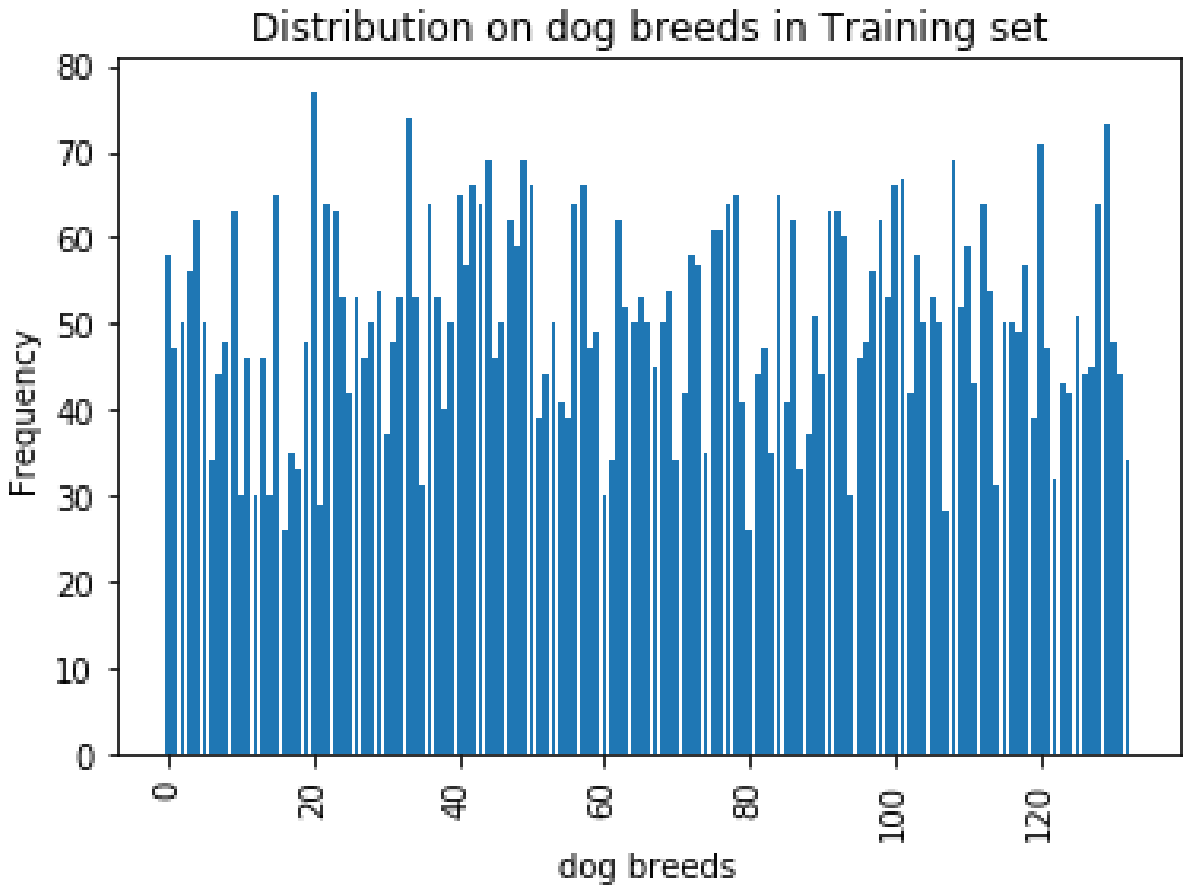


Exploratory Visualization

We can plot out the size of the dog file in a scatterplot with height on the y axis and width on the x axis to see the most common size of the dog images.



From this image, we can see that the height and width for the images in dog file are actually skewed to the smaller sides, with relatively fewer exceptions which has more than 1000 in height/width.
We can also see the distribution of dog breeds in training set.



We can see from this image, that the dog file is unbalanced.

Algorithms and Techniques

The algorithm that I would use to tackle this problem is using Convolutional Neural Network (with transfer learning). This can be useful since we can use pretrained models on imageNet. And since the model is pre-trained with dog images as well, the algorithm can be more accurate if we were to implement an CNN from scratch.

First, we use used human files and convert the images to greyscale. Then using face_fascade and detectMultiScale, we can detect the human faces in the image.

We use VGG16 model to detect dog images. We would crop the image to 244*244 size and use pretrained VGG16 model to detect the dog images. Then using VGG16 model and CNN, we would use pretrained CNN (with last layer modified to output the desired size) and classify the dog breeds accordingly.

Since the CNN is always a very power tool in image classification, and since our training data size is only less than 7000, it would be more reasonable to use a pretrained model. Since VGG16 has always performed well with image classficiation problems, I choose this technique to do the implementation.

Benchmark

For the benchmark, I choose 60% accuracy to be the benchmark of dog breed classifier. Because there are 133 dog breeds in the training set, so a random guess model should have an accuracy of less than 1%. We also build a CNN model from scratch, which has 12% accuracy. Therefore 60% should be a good indicator that our model works as expected.

Methodology

Data Processing

For human file detector, the OpenCV face detector takes the image path, open the image and transform the image into greyscale data.

For the dog image detector, since the dog images are of different sizes, so it's important to standardize the format of the dog images. Because I use VGG16 model in order to detect dog images, I decided to take the input of an image path, open the image, and resize it into 244*244 format, since that's the input size of Karen Simonyan & Andrew Zisserman's input (2015). This input size, therefore, should be seen as reasonable.

Then for two CNN models that I've used, one built from scratch and the other built with transfer learning, I use the same kind of transformation on the training set. Here's the code that I use for transforming training data:

```
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                  std=[0.229, 0.224, 0.225])
preprocessing_data_train = transforms.Compose([transforms.Resize(256),
                                       transforms.CenterCrop(224),
                                       transforms.RandomRotation(30),
                                       transforms.RandomHorizontalFlip(),
                                       transforms.ToTensor(),
                                       normalize])
```

In this transformation, I normalize the image using the suggested normalization parameters in the Python documents, and then I resize the image to 256*256. I then center crop the image into 244*244 size. I did this in order to transform the data into standardized form, with a size of 244*244, while containing most of the information. I also use random rotation and horizontal flip on the training set. I did this in order to augment the training set, in order to avoid the overfitting problem in the training process.

For the validation set as well as the test set, I did the same resizing transformation, in order to have the same standardized format as the training set. However, I skipped the rotation and flipping. Since validation set and testing set are only used to evaluate the

performance of the model, they should avoid any augmentation in order to reflect the performance as accurate as possible.

Implementation

The implementation of the algorithm can be divided into several parts:

1. Human face detector (already provided by Udacity)
2. Dog image detector
3. Dog breed classifier

For the dog image detector, I use pre-trained VGG16 model. I didn't change any parameters or layers of this model, so here's the architecture of the model:

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=
False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=
False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode
=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode
=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode
=False)
  )
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace)
```

```
    (2): Dropout(p=0.5)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace)
    (5): Dropout(p=0.5)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

As we can see, there are 13 conv2d layers and 3 classifier layers, and the output size is 1000. Because the categories corresponding to dogs are indexed with 151-268 in the dictionary, if the predicted category output is within this range, we will decide that the image contains a dog.

Then for the dog classifier model, after the data preprocessing that's discussed in the previous section, I also use pre-trained VGG16 model. I keep all the previous conv2d layers in the model, and change 3 linear classifier layers at the end to get the output size I need:
model_transfer = models.vgg16(pretrained = True)

```
model_transfer = models.vgg16(pretrained = True)


## Freeze model weights
for param in model_transfer.parameters():
    param.require_grad = False

n_inputs = model_transfer.classifier[0].in_features
model_transfer.classifier = nn.Sequential(nn.Linear(n_inputs, 4096),
                                          nn.ReLU(),
                                          nn.Dropout(0.4),
                                          nn.Linear(4096, 512),
                                          nn.ReLU(),
                                          nn.Dropout(0.4),
                                          nn.Linear(512,133))
print(model_transfer.classifier)
if use_cuda:
    model_transfer = model_transfer.cuda()
```

Here is the architecture of the model:

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=
False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace)
```

```
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=
False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode
=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode
=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode
=False)
  )
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.4)
    (3): Linear(in_features=4096, out_features=512, bias=True)
    (4): ReLU()
    (5): Dropout(p=0.4)
    (6): Linear(in_features=512, out_features=133, bias=True)
  )
)
```

With this model, I trained the training set on this model. For the criterion and optimizer, I use

```
criterion_transfer = nn.CrossEntropyLoss()
optimizer_transfer = optim.Adam(model_transfer.classifier.parameters(), lr
=0.001)
```

In order to choose the best model.

For epoch time, I use 20 since this model is pretty complex, an epoch of 20 should be more than enough to yield an reasonable model with accuracy of more than 60%.

One thing to take note is that I only use batch size = 50. First I tried using batch size of 20, and the resulting model prediction was not good enough. Therefore, I tried using a larger batch, 100, but I got an “out-of-memory” error. Therefore I only use batch size of 50, which gives me predictions that’re accurate enough.

Refinement

When I first built the model, I use CNN built from scratch. I use the following architecture:

```
Net(
  (c1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (c2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (c3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (c4): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (c5): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode
=False)
  (fc1): Linear(in_features=12544, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=256, bias=True)
  (fc3): Linear(in_features=256, out_features=133, bias=True)
  (dropout): Dropout(p=0.3)
)
```

In this model, I use 5 conv2d layers, and another 3 linear classifier layer to get the result that I need. I use the same criterion and optimizer as my final model, and train the model on an epoch of 20. The final validation loss I have is 3.699562, and the test accuracy is only 12%.

I notice the model that I built is not complex enough, with only 5 layers and such. However, my training set is only of less than 7000 images, which is not big enough to train with very complex data, and it might lead to overfitting the training data and not performing well with the validation or test set. Therefore, I use the CNN with transfer learning, take in the pretrained VGG16 model and use it for my final model. I also increase the batch size. The final validation loss I have is .858073, which is much better than my previous model. The test accuracy is 69%, also better than my last model. Therefore I take the latter model as more accurate and use it in my final implementation algorithm.

Results

Model Evaluation and Validation

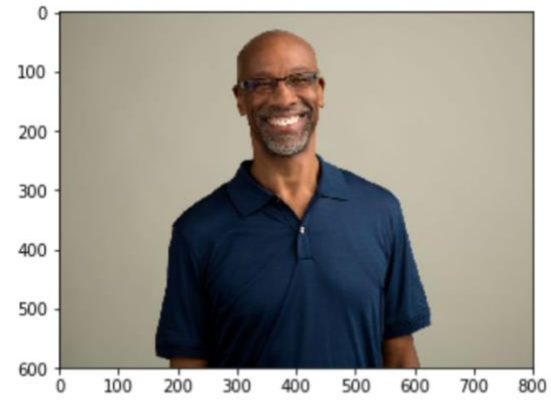
I chose the final model based on the loss on validation set. The final model that I chose has the least loss on validation set. After I did the prediction on the test set, the accuracy percentage gets 69% which is reasonable.

I also did some testing using some images that I found online to test out the performance of the whole algorithm. Based on the result it outputs, it seems that the model can function well enough to detect the human and dog in the image. It can also output pretty accurate predictions on the dog breed. Although the predicted dog breed is not 100% accurate, it seems like it’s doing a great job in general.

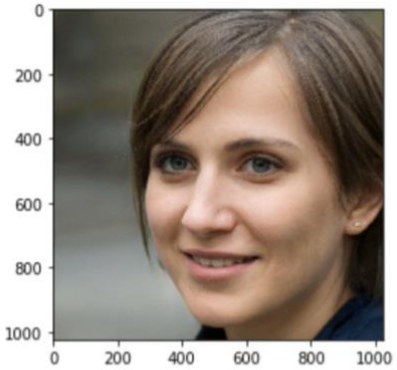
Justification

The final result tested on a very small sample of images shows that the model is performing as expected. It can successfully identify the human or dog in the image, and it can also predict the dog breed relatively well. The reason that the model can sometimes predict the wrong dog breed can be tracked back to the similarities between different dog breed. Another reason can be the model is not trained on enough puppy images, making it harder to predict the right puppy breed.

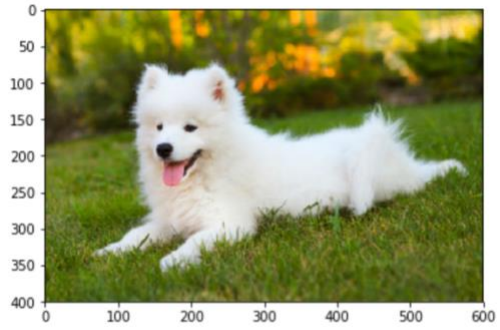
Here're some test images I've tested with the algorithm and resulting output:



```
Hello human!
You look like a ...
 Dogue de bordeaux
```



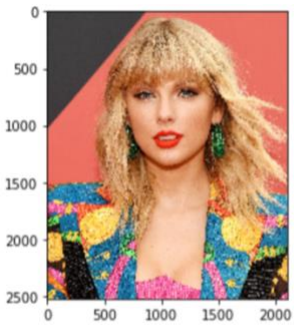
```
Hello human!
You look like a ...
 Yorkshire terrier
```



```
Hello dog!
Your predicted breed is ...
 American eskimo dog
```

```
Hello dog!
Your predicted breed is ...
 Great pyrenees
```

Figure 1 I believe this is a samoyed puppy



```
Hello human!
You look like a ...
 Chinese crested
```

```
Hello dog!
Your predicted breed is ...
 German shepherd dog
```

Although the model can make some mistakes, I believe this is still a good model to use, if the purpose is only to casually identify the dog breed and output resembling dog breed for human faces for fun. It won't be useful enough if we need a model that could accurately (almost 100%) to identify the dog breed.

Conclusion

Reflection

The whole process of the implementation goes as follows:

1. Import the dataset.
2. The data would then transformed into greyscale data, and it will go through the face detector. The model would then determine if the image contains a human face.
3. If not, it will then be transformed into 244*244 size, go through the pre-trained VGG16 model, and determine if the image contains a dog.
4. If the image contains either a human face or a dog, the image will then be normalized and standardized, and the algorithm will use a CNN with pre-trained VGG16 model on the data to determine the dog breed or resembling dog breed.
5. It will then output the sentences stating if the image contains a dog, a human or nothing. It will print out the predicted dog breed if needed.

I think the interesting aspect and the difficult part of the process is to build the dog breed classifier. In that process I tried several models, including CNN from scratch, CNN with transfer learning (with only the first 5 layers included), both of which didn't give me the result that I need. I also adjusted the batch size and epoch size in order to achieve a balance between training time and accuracy, since the final model contains a lot of layers, the training process is extremely slow.

I think the final model and solution fit my expectation for this problem. I think it's okay to use this model to solve the kind of problem discussed in the problem definition.

Improvement

Since my final model only attains an accuracy percentage of 69%, although it's quite reasonable, I believe there are still many room for growth. For example, since the training time for my model is taking too long, I didn't use larger epoch. If I increase the epoch size, it is possible that the model can be more accurate.

Some other pre-trained models, such as ResNet50, or Xception, are something I found during my research on which models to use for image classification. However, due to time limit, I didn't try using those models for my final implementation. However it is still possible that those models can perform better than the current model that I'm using. I would definitely consider using those models in my implementation for improvement.

Since the accuracy is only 69%, using my final model as benchmark, I believe there must be better models which can solve the problem more efficiently and more accurately than what I have right now.