Machine Learning Engineer Nanodegree

Capstone Project

Diyang Wu July 6th, 2020

I. Definition

Project Overview

For the Capstone project that I choose to graduate on the Udacity Machine Learning Engineer Nanodegree, I hope to build an image classification model using Convolutional Neural Network(CNN) on a set of dog images. Computer Vision has been a hot research topic in both academia and industry, with numerous existing well-known pre-trained models such as VGG-16 and ResNet-50. While the performance of these models receives successes and acclaims from app users and companies, I wish to construct such an image classifier myself. The attempt at building a dog image classifier is only a start; the field of Computer Vision has an enormous future in applications and research.

In this specific project, I will try to build the classifier to predict any given image of dogs on its breed. The dataset that I used to train this model comes from AWS and is provided inside of the Udacity framework.

Problem Statement

In this project that I briefly mentioned in the previous section, I wish to construct an image classifier which can predict the breed of a dog. To begin with, note that I have been provided with two datasets, in which one contains the images of dogs, and another contains the images of humans. My goal is to construct an image classifier that can accomplish the following functions:

1. If a dog is detected, the model outputs its predicted breed;
2. If a human is detected, the model outputs his/her predicted resembling dog breed.

The workflow and strategy of achieving our goal can be broken down into the following steps:

1. Exploratory data analysis;

2. Use OpenCV's pretrained face detector to detect human faces;
3. Use VGG-16 to detect dog faces;
4. Create my own algorithm of image classifier, and transfer the architecture under ResNet Model; the final output of this model is the breed of dogs if VGG-16 detects there is a dog in the image, or else the model will output the most resembling dog breed if OpenCV's model detects there is a human in the image.

In the end, I wish to achieve an algorithm on my own of at least 10% test accuracy. I wish to achieve an algorithm using transfer learning under ResNet of at least 60% test accuracy.

Metrics

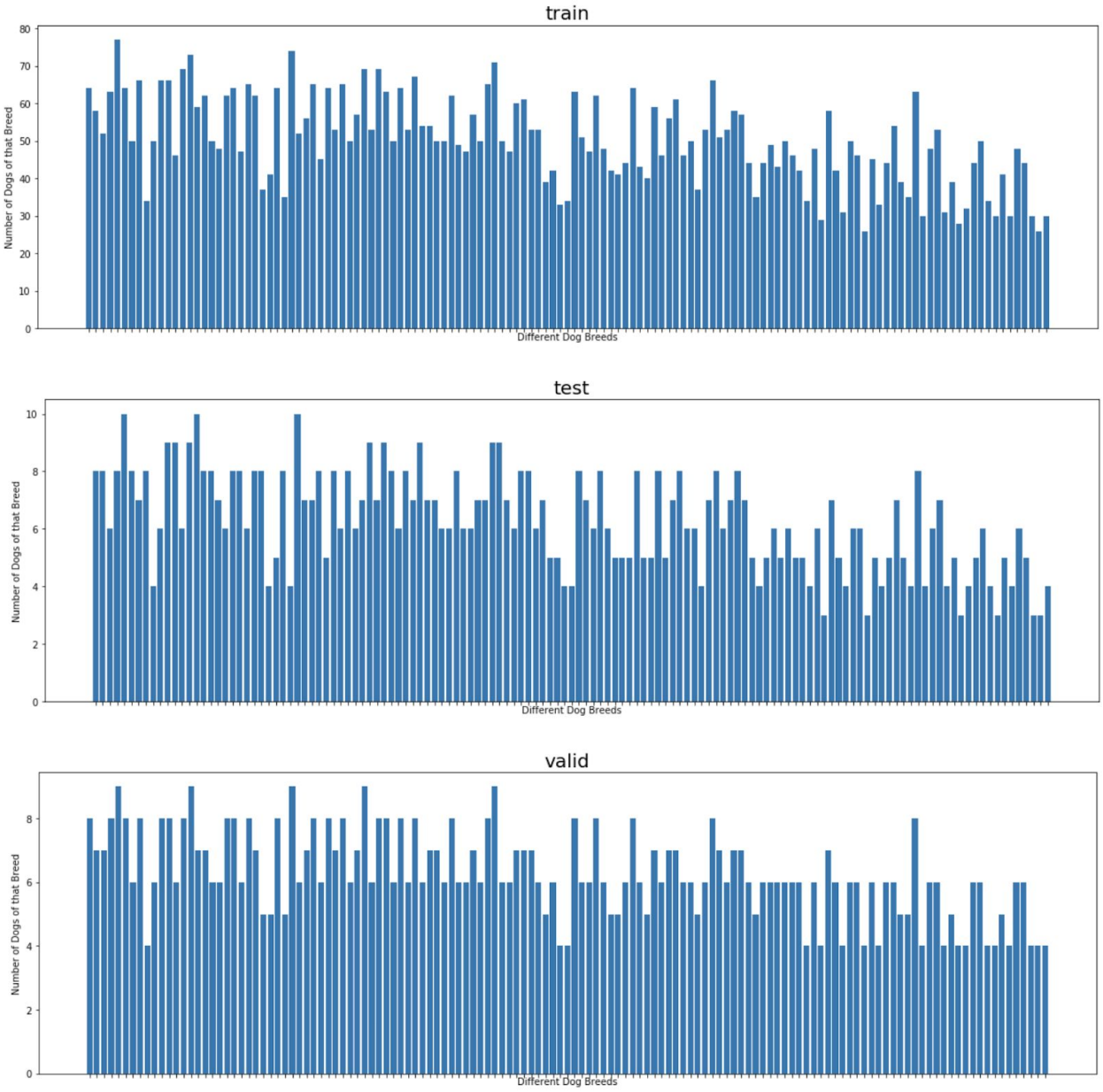
The main evaluation metrics that will be accuracy, which is the number of true positives divided by total number of samples. However, in addition to this metric, some other metrics that can also be taken into considerations are: precision, recall, F1-score, AUC score. Each metric focuses on different aspects of the model performances depending on the distribution of the features for the dataset. In short, if we want to find as many true positives as possible, using accuracy will always be a solid choice; however, as we will see in the Analysis section, since the dataset is not severely imbalanced we will only consider using accuracy this time.

II. Analysis

Data Exploration

The datasets that will be used for this project are from two sources: one contains dog images and another one contains human images. The dog images dataset contains in total 8,351 images with 133 different dog breeds. The human images dataset contains in total 13,233 images. The dataset is provided to me by Udacity, which I assume that under the system of Udacity framework I can use it freely.

Since the specific features of an image dataset are hard to describe, the data will be presented visually dependent on the breed information. Below are the bar plots of the number of dogs in different dog breeds under train, test and validation sets.



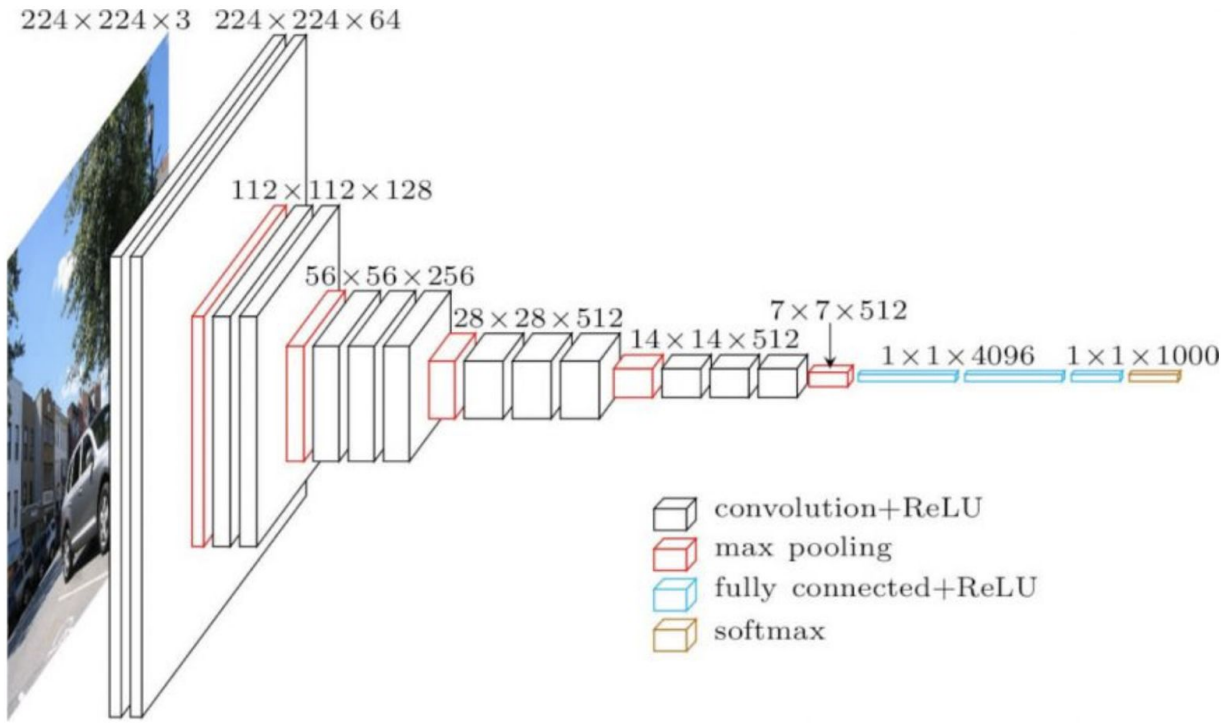
Graph 1. Number of dogs under different breeds

From the above graphs, we can see that even though the distributions of number of dogs under different breeds are not perfectly uniform, it's safe to say that they are not severely imbalanced. Therefore we do not need to handle the data with other sampling methods, and we do not need to incorporate other metrics to justify our model performance.

Algorithms and Techniques

The two specific pretrained models that we used outside of our own algorithms are VGG-16 and ResNet50. Below I will introduce these two models in details.

VGG-16:



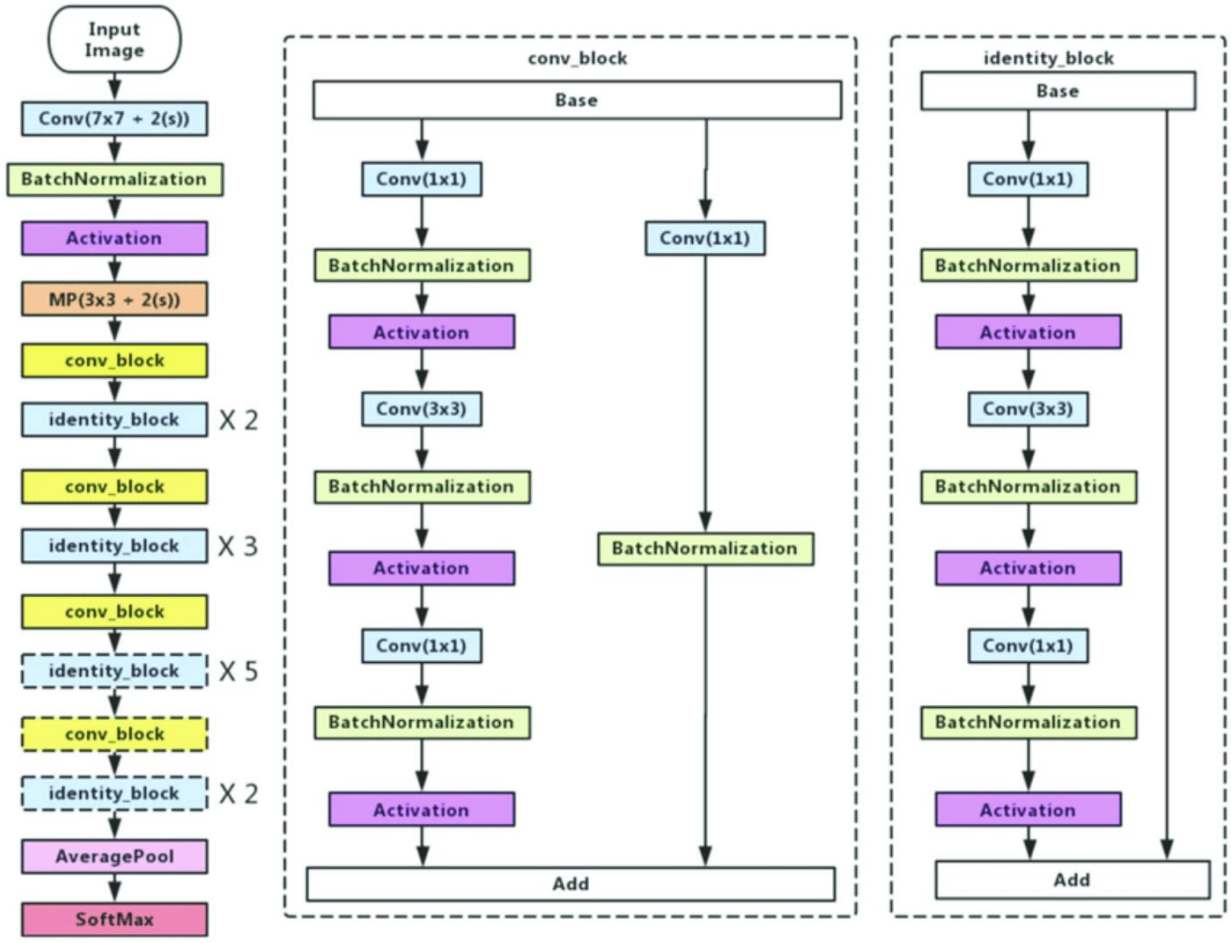
Graph 2. VGG Model Architecture

Source: https://neurohive.io/en/popular-networks/vgg16/

VGG-16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition”. Above shows the numerous layers that VGG-16 has and the number of parameters at each stage of the model. The remarkable results of VGG-16 have brought attention from across different industries, with 92.7% test accuracy, which later became the top 5 best performing models in ImageNet.

In my project, VGG-16 will serve as a “pre-filter” of my image; that is, VGG-16 will first detect if the given image is an image of a dog.

ResNet50:



(Left) ResNet50 architecture. Blocks with dotted line represents modules that might be removed in our experiments. (Middle) Convolution block which changes the dimension of the input. (Right) Identity block which will not change the dimension of the input.

Graph 3. ResNet50 Model Architecture

Source:

https://www.researchgate.net/figure/Left-ResNet50-architecture-Blocks-with-dotted-line-represents-modules-that-might-be_fig3_331364877

ResNet is a short name for Residual Network. As the name of the network indicates, the new terminology that this network introduces is residual learning.

In general, in a deep convolutional neural network, several layers are stacked and are trained to the task at hand. The network learns several low/mid/high level features at the end of its layers. In residual learning, instead of trying to learn some features, we try to learn some residual. Residual can be simply understood as subtraction of features learned from input of that layer.

ResNet does this using shortcut connections (directly connecting input of n-th layer to some (n+x)th layer. It has proved that training this form of networks is easier than training simple deep convolutional neural networks and also the problem of degrading accuracy is resolved.

Thus ResNet50 has served as the backbone of many important deep learning frameworks. In my project I will do exactly the same: I will use ResNet50 as the base of my transfer learning. I will utilize the existing weights from ResNet50, which has trained on millions of images, and change the last output fully connected layer to cater to my desirable output.

Benchmark Model

For this model, the benchmark CNN model is the model that I created myself. Since there is in total 133 dog breeds, making the chance of guessing a right dog breed lower than 1%, I will expect my own algorithm reaches at least 10% accuracy.

```
Net(
  (layer1): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (layer3): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc_layers): Sequential(
    (0): Dropout(p=0.2)
    (1): Linear(in_features=6272, out_features=500, bias=True)
    (2): ReLU()
    (3): Dropout(p=0.2)
    (4): Linear(in_features=500, out_features=133, bias=True)
  )
)
```

Graph 4. My Model Architecture

As shown above, I created three convolutional layers, with ReLU activation function for each layer and Max Pooling at the end. For the final fully connected layers, I used two, one for global

average pooling and one for dense layer.Notice that the input for the images is 224x224, and after normalization.

III. Methodology

Data Preprocessing

Since the datasets that are given to me are already cleaned, clearly labelled, and splitted into train, test and validation sets, the only left for me to do is to transform the image data to something that can be used by the model. Below summarizes the step to reach that goal.
For all three datasets I have different coping method in terms of resizing the images, and the reason is as followed:

1. For the train set, I randomly resize the image to 224x224, because we want as much variation in the train set as possible. By doing that our classifier can hopefully recognize as many patterns as possible. This is also the reason why I use random horizontal flip in the images from the train dataset.
2. For the validation set, I resize all images to 256x256, and did a center crop for the image to be 224*224, because at the validation step I want my results to be comparable to each other. That way if overfitting happens, I can stop the training at an early stage.
3. For similar reasons to the validation set, I resize the image to 224*224 in the test set, because I want a comparable result.
4. For all of the above sets, I implemented the normalization step as suggested by the official PyTorch documentation. The parameters for normalization are mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225].

Implementation

After all the data preprocessing work, I create a CNN architecture as shown above. The result is evaluated by the number of correct predictions in test data divided by the total number of test data. Since the model is implemented under the framework of PyTorch, it is extremely easy for beginners like me to design a convolutional neural network, so it’s safe to say that I did not encounter any significant coding hardship during the process. The specification for the model parameters is set as followed:

1. 3 Convolutional layers, with ReLU activation, and MaxPooling. 2 Fully Connected layers, with ReLU activation;
2. The first convolutional layer has 32 channels, the second layer 64 channels and the third layer 128 channels;
3. Mini Batch Size: 30;
4. Loss Function: Multi-Class Cross Entropy;
5. Optimizer: Adam Optimizer with 0.001 Learning Rate.

The final model that's used in the run_app function is the model created out of transfer learning. Basically I used the already trained ResNet model, along with its weights, and transferred the last layer to a fully connected layer to output my desired dimension. Thus the only training that should be done is the last layer; the other layers should remain unchanged. The specific parameters for the model is as followed:

1. Original structure of all the layers from ResNet50 except for the last layer. 1 Fully Connected layer at the end with dimension (2048, 133);
2. Loss Function: Multi-Class Cross Entropy;
3. Optimizer: Adam Optimizer with 0.001 Learning Rate.

Refinement

During the training process of the two models above, I experimented with different sets of numbers of training parameters. Specifically, I tried a different number of training epochs, different learning rates, and different optimizers. To be specific, the ranges that I tried for each parameter are:

1. Learning Rate: [0.1, 0.01, 0.001, 0.0001]
2. Epoch: [10, 20, 30, 40, 50]
3. Optimizer: [SGD, Adam]

In general, the lower the learning rate, the higher the epoch, and Adam optimizer perform the best during the training phase. If considering the overfitting problem, one might argue that a learning rate of 0.001, training epoch of 40 and optimizer of Adam will be the best combination.

IV. Results

Model Evaluation, Validation and Justification

For the final model that I described in the previous section, I achieved 82% accuracy on the test set. Compare this with the initial model performance, which is 17%, I had a 382% improvement using transfer learning. Just from looking from the results, I can justify that my final model is significantly better than the benchmark model.

To remind you again with my model architecture, please see as followed:

1. Original structure of all the layers from ResNet50 except for the last layer. 1 Fully Connected layer at the end with dimension (2048, 133);
2. Loss Function: Multi-Class Cross Entropy;
3. Optimizer: Adam Optimizer with 0.001 Learning Rate.
4. For all of the datasets, I implemented the normalization step as suggested by the official PyTorch documentation. The parameters for normalization are mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225].

To test the robustness of this model, I did some adjustments and added some noises, such as random horizontal flips in our training sets. However, our model performs consistently in all these situations, further proving that our model is robust to noises.

Model Improvements

After every project, it is always important to reflect back on what could be improved. Here I think about three ways that I can further improve the models.

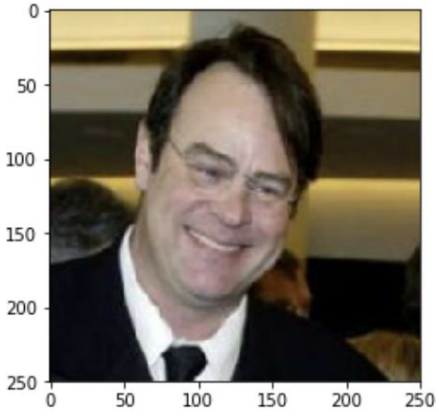
1. One of the biggest problems during the training process is that there is not enough data. Take an example from my personally created network, if the data is significantly bigger, I think the model performance would not be as low as 20%.
2. Decreasing learning rate and increasing number of epochs might also help to improve the accuracy of the model. It is true that we might encounter problems of overfitting, but then again, with big enough data, if we can get an early stop in the validation set and see the performance in the test set, overfitting could be diagnosed.

3. Using cloud computing for the training/testing process. Using the GPU mode under Udacity framework has already given me a boost of speed for training; however, if I can further utilize AWS, I might be able to train more data with less time. More training will likely improve the model accuracy.

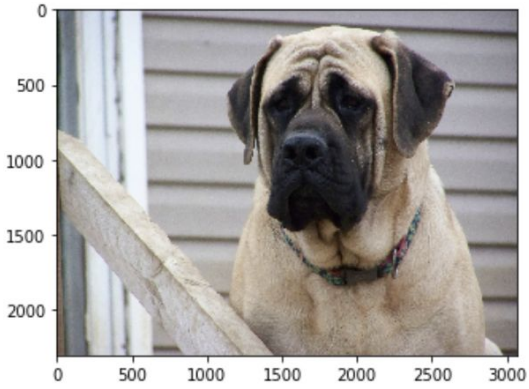
Sample Model Output

At the end of this project I want to give you some sample output from our models. The sample output includes some of the pictures from the original datasets, as well as some of my own pictures. Hopefully you can enjoy them!

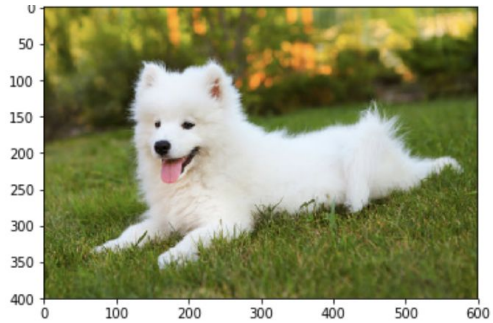
```
hello, human!
```



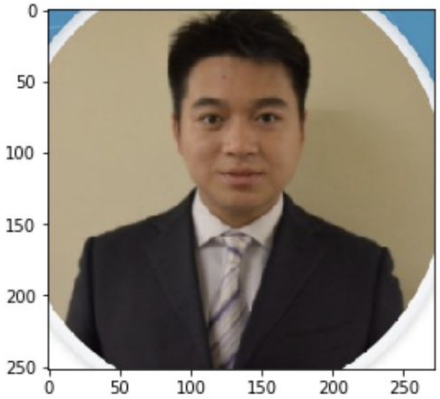
```
You look like a ... Chihuahua
```



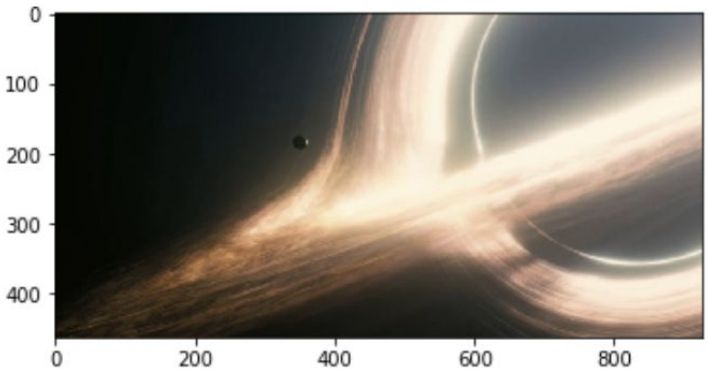
```
The cutie in the picture is a ... Mastiff
```



```
The cutie in the picture is a ... American eskimo dog
```



```
You look like a ... Australian shepherd
```




```
There is no human or dog in the picture!
```


```
There is no human or dog in the picture!
```