Machine Learning Engineer Nanodegree Capstone Project

CNN Project: Dog Breed Classification

Burak Kılıçaslan

October 21th, 2020

I. Definition

Project Overview

In today's world, the subject of artificial intelligence is very popular and AI applications are followed by everyone. Researches and academic studies continue rapidly on this field to find new use-cases. One of the most popular AI algorithms is deep learning in recent years. With development of hardware technologies and deep learning algorithms, image and video processing applications become so popular and useful for many people and many companies. However, apply these algorithms and techniques are so challenging and complicated problems in real world.

One of these challenging applications is Dog breed classification. In this application, there are 133 different classes and all of them belong to dogs. Classifying different species such as dog and horse is relatively easy comparing to classifying same species breeds. For this reason working such this project and getting high performance is a tough problem. For image classifying or object detection applications, there are lots of different concepts and algorithms in literature to tackle with these problems. In this project, we will use Convolutional Neural Networks which is the state of art for image or video processing.

CNN provides better feature extraction for complicated problems. Its architecture is similar human brain. It contains lots of layers and lots of neurons in every layer. Every layer’s neurons are connected with next layer’s neurons and in training phase, this connection between neurons keep information about data or for our project images. This architecture allows to us build complicated image processing applications with high accuracy. On the other hand, this architecture contains lot of computational operation and this means that it needs high hardware performance. As we stated above, thanks to development on hardware technologies, GPU performances and cloud services, it is easy to provide requirements for this situation.

To sum up, in my Udacity Machine Learning Nanodegree Capstone Project, I follow the steps CNN architectures and try to create CNN model from scratch. This challenging project gives me chance to face difficulties on image classification applications and gain knowledge from these difficulties.

Problem Statement

In this project, we build a pipeline to process real-world images. Provided an image by user, our algorithm will detect that, the image is human or dog. If image is dog than our algorithm will predict its breed. If the image is human than our algorithm will predict which breed of dog the human image looks like.

The project steps are as below;

- Step 0: Import Datasets
    - Because we will use udacity workspace, we will not need to download dataset in our local machine. Dataset has been already uploaded by Udacity in the workspace. We just need to load and read data from workspace folder.
- Step 1: Detect Humans with OpenCV Haar feature-based cascade classifier
    - We will use the OpenCV Haar feature-based cascade classifier which is explained detailed on solution statement section. We will import this classifier using OpenCV and give outputs on human images.
- Step 2: Detect Dogs with pre-trained VGG-16 model
    - Firstly, we will use VGG-16 pre-trained model to detect dog breeds. We just load the model to our workspace and using the model, test our images.
- Step 3: Create a CNN to Classify Dog Breeds (from Scratch)
    - This is one of the most challenging part in the project. CNN models are so complicated and we will try to build a CNN model from scratch and train this model with our dataset. We will try to reach at least %10 accuracy and to do that we should tune our hyper parameter and model architecture.
- Step 4: Create a CNN to Classify Dog Breeds (using Transfer Learning)
    - Because getting high accuracy results with creating and training a CNN model from scratch is really tough problem, we will use transfer learning which is that training the model with using a pre-trained model weights and features. So, we can increase our accuracy rate.
- Step 5: Write final Algorithm
    - We will write an algorithm which takes the images from users and gives the results. This algorithm contains everything that we need. It will read and load the image that user upload, calls required functions to identify objects, based on the result decide what it prints.
- Step 6: Test final Algorithm and results
    - Testing final algorithm and whole project codes.

Metrics

Accuracy metric will be used on all steps of this project. Besides, Udacity reviewers evaluate my project according to accuracy metric as mentioned above. Because our dataset is balanced fairly, accuracy metric can be used for evaluation.

The expected accuracy result for CNN from scratch should be at least %10.

The expected accuracy result for using transfer learning should be at least %60.

II. Analysis

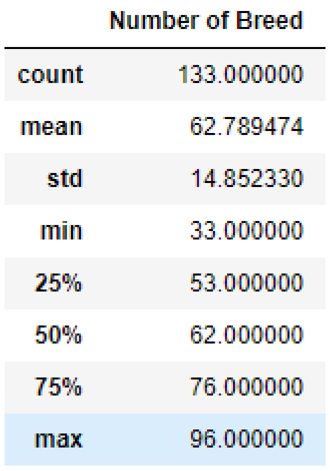
(approx. 2-4 pages)

Data Exploration

The dataset has been provided by Udacity. Dataset contains 13233 human images and 8351 dog images, which are separated as train, test and validation. Human images or lfw size is 180 MB. Dog image's size is 1.1 GB. Dog images have 133 different class and it is already labeled by Udacity. On the other hand, human dataset is ordered according to name.

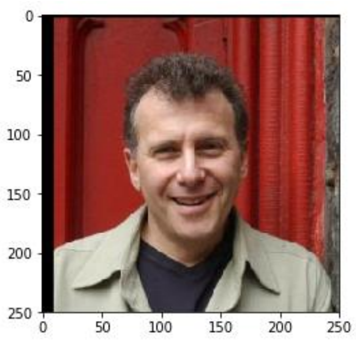
When we look into datasets size, size of human images is constant as (25,250,3) but size of dog images differ from each other. You can see histograms as below.

As we said above, the dataset has 133 different classes. When we look into dataset more deeply, there are 62 number of image for every dataset on average. 'Norwegian Buhund' breed class has 33 number of images that are min value between all classes and 'Alaskan Malamute' has 96 number of images that are max value between all classes. You can see detailed information about distribution of the dataset as below.



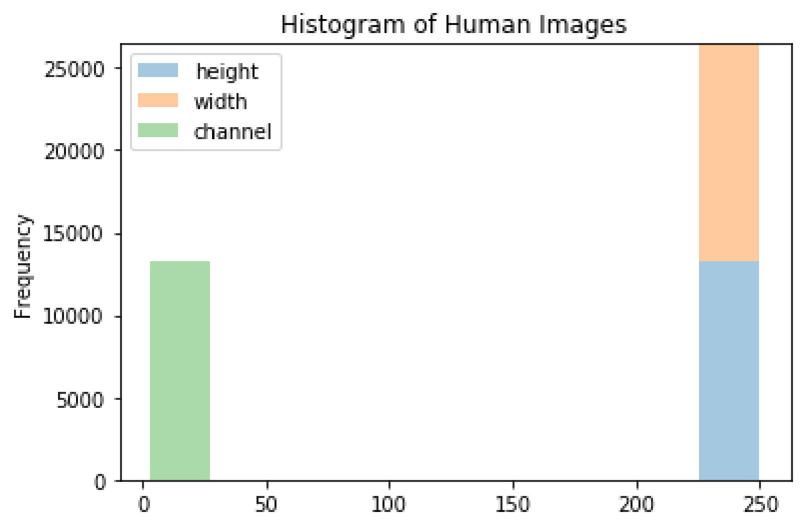
	Number of Breed
count	133.000000
mean	62.789474
std	14.852330
min	33.000000
25%	53.000000
50%	62.000000
75%	76.000000
max	96.000000

Example images;

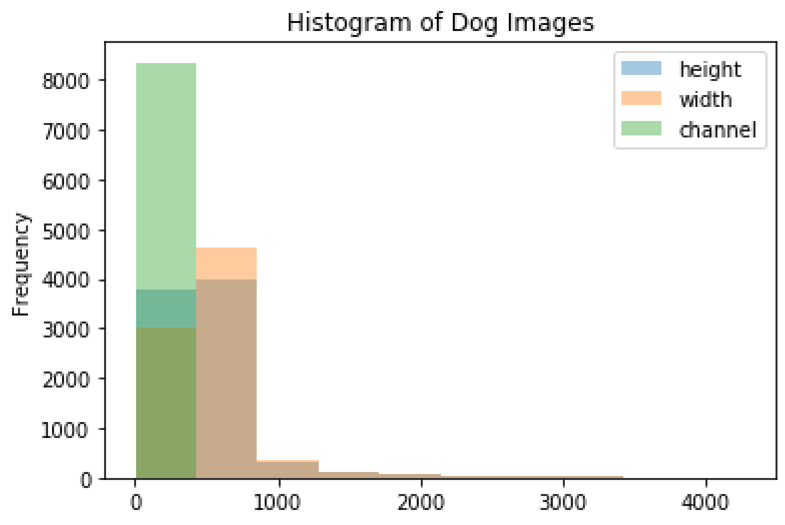


Exploratory Visualization

We can see histograms of human images and dog images to prove distribution of their sizes.



Histogram of human images proves that all images have same height and width. To understand clearly that, dataset has same size of width and height, I plotted this histogram as stacked.

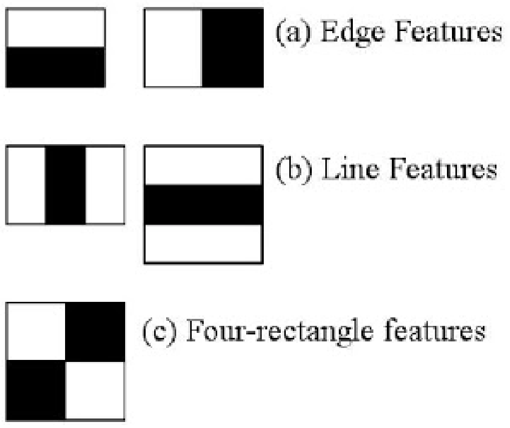


Histogram of dog images proves that images have different height and width.

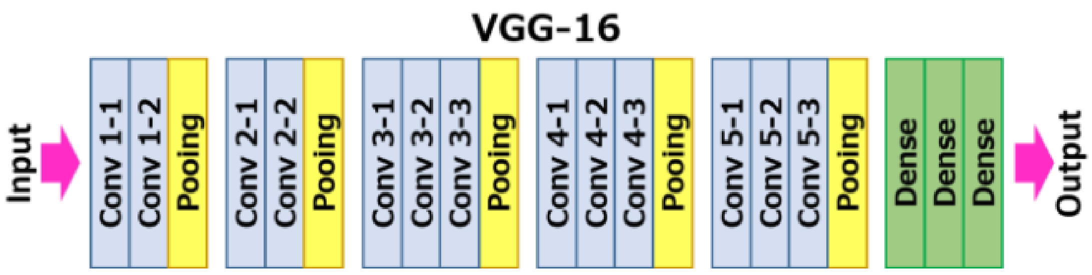
Algorithms and Techniques

In this project, firstly, we detected the image was human or dog. We used openCV's Haar feature-based cascade classifiers for human detection and pre-trained VGG-16 model for dog breeds classifying.

OpenCV haar cascade classifier and is proposed by Paul Viola and Micheal Jones in 2001. It is a machine learning method for object detection and used as effectively. It needs lots of true and false images to train. If we create a face detection than true images are images with face, false images are images without face. After that, feature extraction process starts from these images with sum of the pixels of the image location by location. Feature extraction processes based on three features which are line, edge and four-rectangle as you can see below clearly. Because calculating features for all possible locations need so much computation and there may be lots of irrelevant features on an image, Adaboost algorithm which is a learning algorithm to improve performance uses during feature extraction process. After feature extraction, best features are selected according to minimum error rate and classification process start. After every classification, based on the result weights are updated and this loop continue until model converge with optimum metric.



VGG-16 model is a simpler CNN architecture. It contains 16 layer which last three of them fully connected (dense) layer. It contains about 138 million parameter computation and gives output from dense layer about 1000 classes which is calculated with softmax activation function. Softmax activation function gives output between 0 and 1, so it uses as very popular for classification problems. Inputs of this model 224x224 pixel. You can see the model architecture clearly as below.



In this project, we created a CNN model from scratch and trained our model. When we trained our model, we used four convolutional layer and three fully connected layer

The algorithm that I used in the final was explained as below;

1. The algorithm takes the images from users and send to our human detector and dog detector.
2. If human detector is true, than the algorithm sends the image to dog_breed_classifier and gives the results.
3. If dog detector is true, than the algorithm sends the image to dog_breed_classifier and gives the results with breed of dog.
4. If both dog detector and human detector is not true, than algorithm prints error message that is “no dog or human can be found”.

Benchmark

The first task in this project was that creating CNN model from scratch. This model should be at least %10 test accuracy.

After our CNN model, we used transfer learning to improve our CNN. The improved model test accuracy should be at least %60.

Actually %10 accuracy is very low however image classification is challenging and hard task. It needs so many data and very complex model with long training sessions. We can see that it is so complicated to get high accuracy results immediately with creating basic CNN models.

III. Methodology

Data Preprocessing

Data augmentation is the critical preprocess for image processing as my researches to prevent overfitting and get more generalized images. It makes our model has higher accuracy with augmented inputs. Hence, I applied some data augmentation techniques.

Because our dog images have different size, we have to resize them before feeding our network. I resized them as 250*250 and crop as 224*224. So our images size for input tensor became 224*224 because it is default input size for VGG16 models. Resizing image as 250*250 was the important preprocessing for my model. I think that is because images have different size in dog dataset and when we resize as 250*250, we standardize width and height then crop so we did not lose any important subject in the image.

Besides of them, I applied random rotation and random horizontal flipping. At the end, Data normalization is applied as the final step to our data to make similar data distribution on our training dataset.

Summary of all preprocessing steps are as below;

- Resize the image as 250*250

- Apply random rotation with 30
- center crop the image as 224*224
- Apply random horizontal flip
- Convert to tensor
- Normalize the data as ([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]) because most of models are used this normalization values.

I did not apply any data augmentation techniques on test and validation data except resizing and cropping. After that, I normalized them as well.

Implementation

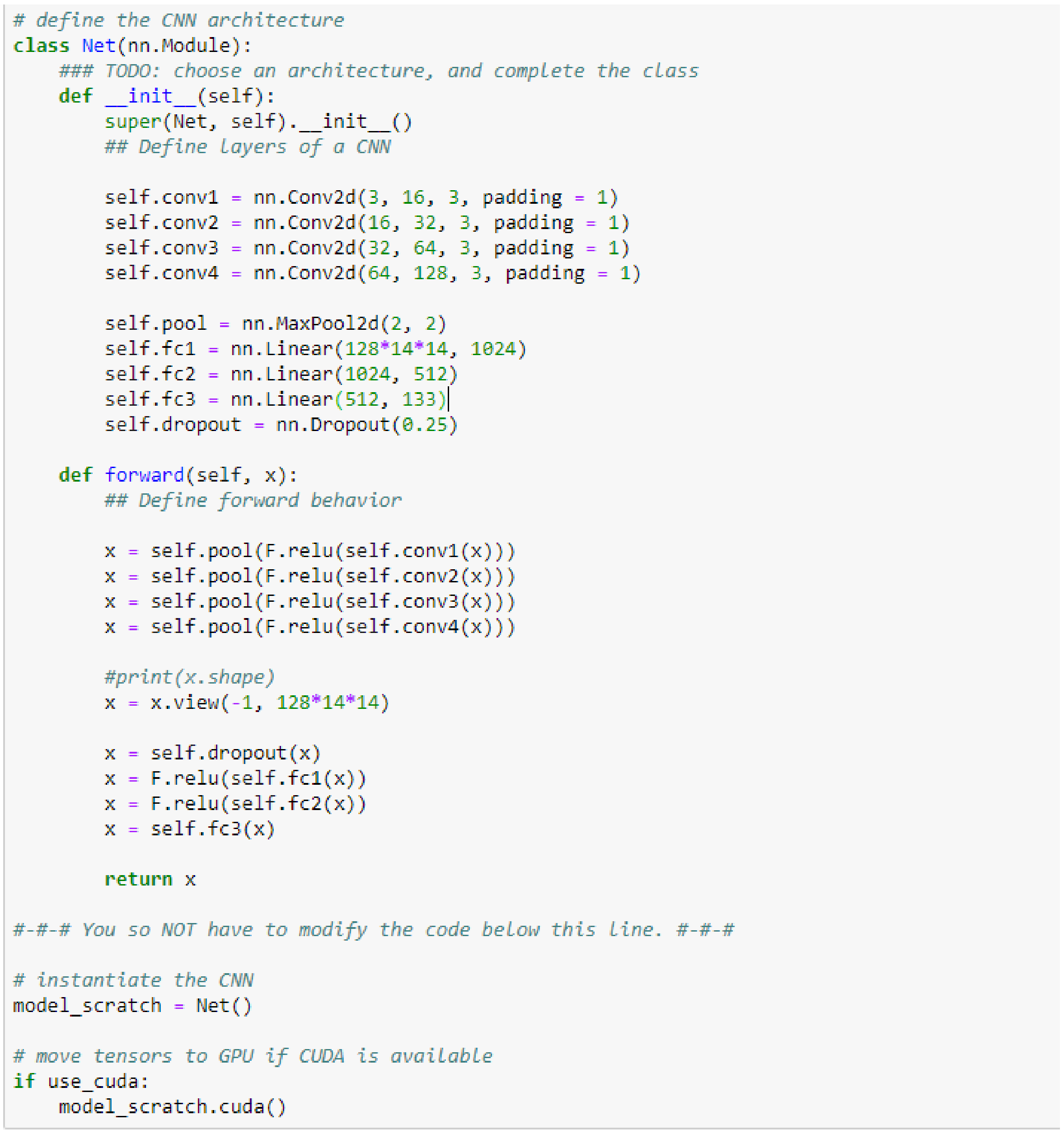
To finalize this project, five steps is completed which are as below;

1. Human detector
2. Dog detector
3. Create a CNN from scratch to classify dog breeds
4. Create a CNN Using transfer learning to classify dog breeds
5. Final algorithm to use these models.

In the first step, udacity provided required codes and algorithm with using OpenCV haar feature based cascade classifier, which I explained above as detail.

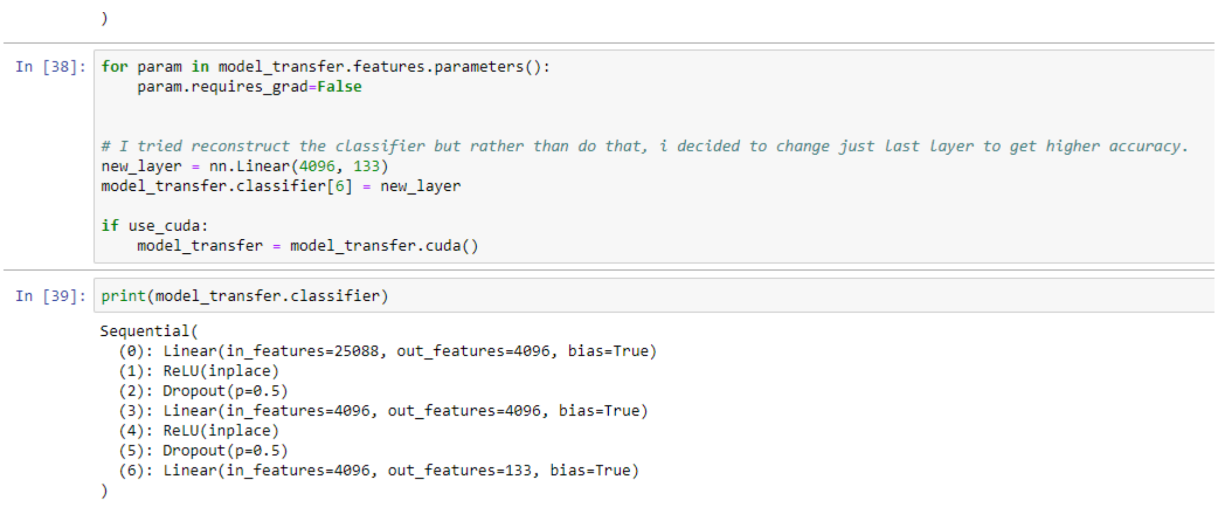
In the second step, I used pre-trained CNN model VGG16 to detect dogs.

The third step was the one of the main part in this project. Creating CNN model from scratch and training them is challenging issues. I had to make lots of search to create working model and understand theory behind of the codes. I used pytorch to build CNN model and udacity workspace because of GPU support. After preprocessing of my input data, I build with four convolutional layer. These convolutional layers have 3x3 kernel size, padding of 1, stride of 1 and max pooling layer with 2x2 window size. RELU activation function, which is the state of the art was applied after each convolutional layer. After four convolutional layer, we have feature maps with 14*14*128 tensor shape. We flattened output of feature maps and applied three fully connected layer for classification. First fully connected layer took 25088 vector size as input and gives 1024 vector size as output. Second fully connected layer took 1024 vector size as input and 512 vector size as output. Last fully connected layer took 512 vector size as input and 133 vector size as output which is our dog breed classes. RELU activation function was applied after each fully connected layer as well except last layer. To make classification, loss criteria was set as cross entropy and Adam optimizer was selected because of it is powerful optimizer and state of the art in literature. Code snippet below shows the architecture of my model and code implementation.



```
# define the CNN architecture
class Net(nn.Module):
    ### TODO: choose an architecture, and complete the class
    def __init__(self):
        super(Net, self).__init__()
        ## Define layers of a CNN

        self.conv1 = nn.Conv2d(3, 16, 3, padding = 1)
        self.conv2 = nn.Conv2d(16, 32, 3, padding = 1)
        self.conv3 = nn.Conv2d(32, 64, 3, padding = 1)
        self.conv4 = nn.Conv2d(64, 128, 3, padding = 1)

        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(128*14*14, 1024)
        self.fc2 = nn.Linear(1024, 512)
        self.fc3 = nn.Linear(512, 133)
        self.dropout = nn.Dropout(0.25)

    def forward(self, x):
        ## Define forward behavior

        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = self.pool(F.relu(self.conv4(x)))

        #print(x.shape)
        x = x.view(-1, 128*14*14)

        x = self.dropout(x)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)

        return x

#-#-# You so NOT have to modify the code below this line. #-#-#

# instantiate the CNN
model_scratch = Net()

# move tensors to GPU if CUDA is available
if use_cuda:
    model_scratch.cuda()
```

In the fourth step, I selected VGG16 pre-trained model for transfer learning. I froze all feature parameters. I just changed the last fully connected layer output as 133 and trained classifier again. In this model, cross entropy loss criteria was selected because of making classification and Adam optimizer was selected. Code snippet below shows the architecture of my model and code implementation.

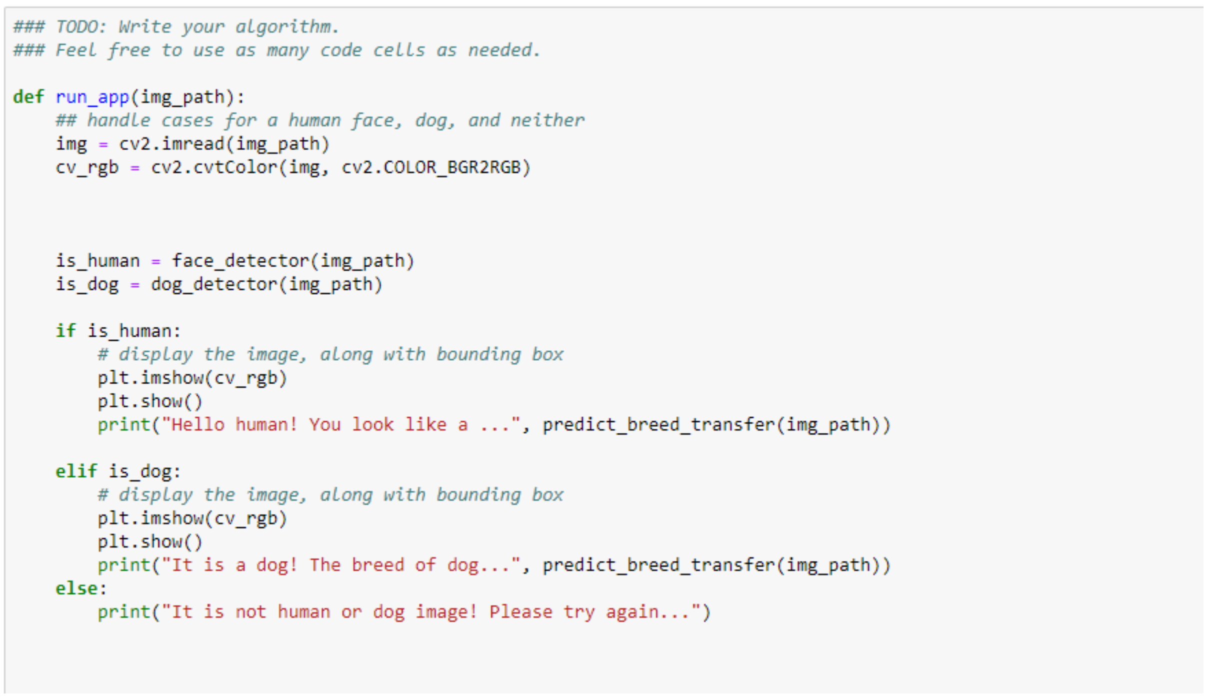


```
        )
```

```
In [38]: for param in model_transfer.features.parameters():
             param.requires_grad=False


         # I tried reconstruct the classifier but rather than do that, i decided to change just last layer to get higher accuracy.
         new_layer = nn.Linear(4096, 133)
         model_transfer.classifier[6] = new_layer

         if use_cuda:
             model_transfer = model_transfer.cuda()
```

```
In [39]: print(model_transfer.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=133, bias=True)
         )
```

In final step, the algorithm takes images and send them to dog detector and human detector functions. If human detector function returns true then it prints result and sends to images to "predict_dog_breed" function to estimate which breed that human looks like. If dog detector returns true then it prints result and sends images to "predict_dog_bree" function to estimate which breed that dog is in the image. Code snippet below shows the architecture of my model and code implementation.



```
### TODO: Write your algorithm.
### Feel free to use as many code cells as needed.

def run_app(img_path):
    ## handle cases for a human face, dog, and neither
    img = cv2.imread(img_path)
    cv_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)



    is_human = face_detector(img_path)
    is_dog = dog_detector(img_path)

    if is_human:
        # display the image, along with bounding box
        plt.imshow(cv_rgb)
        plt.show()
        print("Hello human! You look like a ...", predict_breed_transfer(img_path))

    elif is_dog:
        # display the image, along with bounding box
        plt.imshow(cv_rgb)
        plt.show()
        print("It is a dog! The breed of dog...", predict_breed_transfer(img_path))
    else:
        print("It is not human or dog image! Please try again...")
```

Refinement

The both model gave the poor results in the first trial. With more research on the internet and previous capstone projects, I improved my model with some tuning on parameters and architecture. I summarized below table with comparing results of my progress. Which parameter changed for every step was highlighted as green.

#	CNN layer	Fully Connected Layer	Optimizer	Loss Criteria	Epochs, LR, Batch Size	Transformation techniques	Dropout	Accuracy
1	3 CNN Layer	2 Dense Layer	SGD	Cross Entropy	Epochs:20 LR: 0.01 Batch S.: 32	Random Rotation, Resized Crop, Horizontal Flip, Normalize as 0.5	No	%0
2	4 CNN Layer	3 Dense Layer	Adam	Cross Entropy	Epochs:20 LR: 0.001 Batch S.: 128	Random Rotation, Resized Crop, Horizontal Flip, Normalize as 0.5	Yes, 0.25	%10
3	4 CNN Layer	3 Dense Layer	Adam	Cross Entropy	Epochs:20 LR: 0.001 Batch S.: 64	Random Rotation, Resize as 256, Crop as 224, Horizontal Flip, Normalize as [0.485, 0.456, 0.406], [0.229, 0.224, 0.225]	Yes, 0.25	%0 (Overfittig)
4	4 CNN Layer	3 Dense Layer	Adam	Cross Entropy	Epochs:20 LR: 0.001 Batch S.: 128	Random Rotation, Resize as 256, Crop as 224, Horizontal Flip, Normalize as [0.485, 0.456, 0.406], [0.229, 0.224, 0.225]	Yes, 0.25	%0 (Overfittig)
5	4 CNN Layer	3 Dense Layer	Adam	Cross Entropy	Epochs:20 LR: 0.001 Batch S.: 128	Random Rotation, Resize as 256, Crop as 224, Horizontal Flip, Normalize as 0.5	Yes, 0.25	%13

Model architecture and hyper parameters that I choose are above table. According to my selections, progress of accuracy was very good. First of all, I realized that Adam optimizer is really powerful and the best option for this problem. The other important point was that how much deep our model and channel number in CNN layers. When I increased layer of my model, accuracy increased as well. In my first model, learning rate was 0.01 and this was a high number like these problems. So I changed learning rate as 0.001 in my other models and this affected accuracy positively. I have also realized that Batch size was crucial hyper parameter for deep learning algorithms. Choosing larger batch sizes causes to overfitting because of trying to converge completely. In the beginning, I chose lower batch sizes but the next steps I decided to select batch sizes as 128. Normalizing images were the other important step on my model, which ensures each input parameter has similar data distribution. When I normalized my data same as VGG-16 pre-trained model, my accuracy was %0 because of overfitting but I normalized my data as 0.5, it prevented overfitting and I had higher accuracy.

In CNN with transfer learning, I used pre-trained VGG-16 model. I did not train feature extraction parameters layers. I just build classification layer and train this layers to make classification with 133 classes different from VGG-16 model. I changed the last layer output as 133 and train gradients in these layers again with our images. Transformation techniques on the data effected my model accuracy for this model in my case. I applied same transformation techniques in the beginning but my accuracy was so bad as %26. After I applied different transformation with using resize and crop rather than random resized crop and my accuracy increased very well as %70.

IV. Results

Model Evaluation and Validation

Accuracy results proved that using transfer learning was clearly better choice to make classification in the final application. To reach this accuracy, I tried different options and picked the best values according to results. I keep epoch number constant as 20 for my every trial. In every epoch, validation accuracy was compared with previous epoch validation accuracy and if it is lower than previous epoch, model was saved automatically. Thus, we had the best model given the lowest accuracy in that training. The model was tested with different images that was not in train, test and validation dataset. Both models passed the required accuracy values and benchmark metrics.

Justification

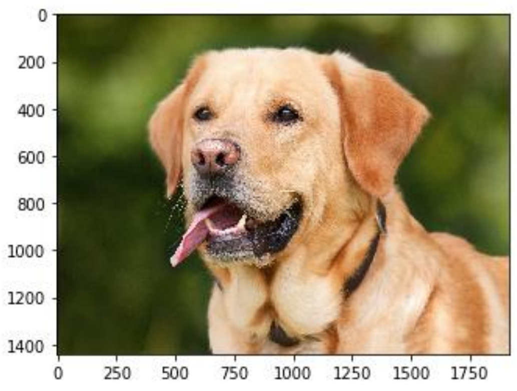
The required accuracy was min %10 for CNN from scratch. My model reached %13 testing accuracy so requirement met successfully.

The required accuracy was min %60 for CNN using transfer learning. My model reached %70 testing accuracy using VGG16 feature extraction parameters so requirement met successfully as well.

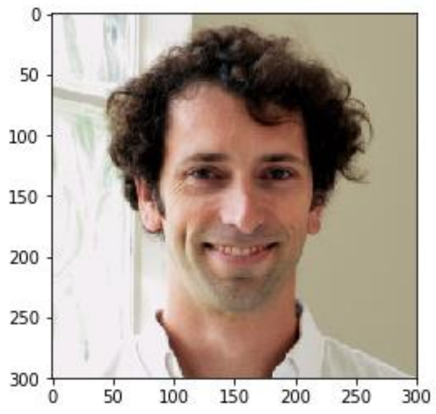
V. Conclusion

Free-Form Visualization

When I tested final application with my own images outputs were resulted as below.

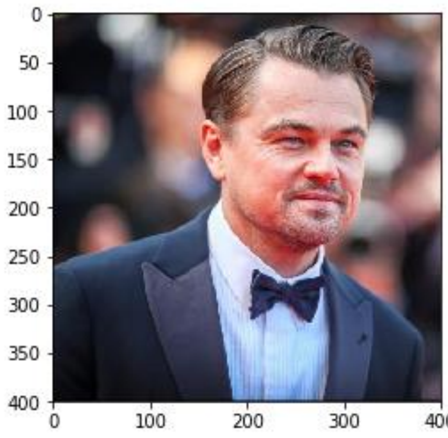


```
It is a dog! The breed of dog... Golden retriever
```



```
Hello human! You look like a ... Xoloitzcuintli
```

```
It is a dog! The breed of dog... American staffordshire terrier
```



```
Hello human! You look like a ... Xoloitzcuintli
```

```
It is a dog! The breed of dog... French bulldog
```

```
It is not human or dog image! Please try again...
```

Reflection

First of all, thank your organization for providing this training and sources. I was really exciting and instructive course. Of course, it was also tiring especially for me because my company was put deadline for this course much shorter than Udacity deadline. However, I really enjoyed learning in this journey. I had chance to work on real machine learning project and experiment a lot of obstacles during this process. I heard deep learning and read about how it works on few blog posts. I am so happy that I move forward from reading a few blog-posts to use it with coding and learning deeply with research in articles. I realized that machine learning is not pure software engineering, it is much deeper than software engineering and needs mathematic and statistic knowledge as much as software engineering. In the meantime, it provides so much use-cases and applications to make easy people works or life.

Improvement

I could improve my application both in terms of model accuracy and user experience. My model accuracy in transfer learning met expectations with %70 but it was still not enough to use it in real life. Users may want better accurate results. Increasing epoch number and trying different architectures accuracy can be increased. Number of data in training has another key role to get higher accuracy and increasing training data for all classes provides us better models.

In terms of user experience, I could develop a website with flask or another framework. Therefore, users could upload their photos to identify their photos are which breed of dog. It provides more smooth and useful application. Even I could develop a mobile application and it would be perfect uploading photos that application and give the results on the mobile phone.

On the other hand, this application could be enhanced on different dataset to make classification. For example identifying car models or detect masks on the human face would be excellent use case in corona days.

References

1. https://github.com/udacity/machine-learning/blob/master/projects/capstone/report-example-1.pdf
2. https://github.com/udacity/deep-learning-v2-pytorch/tree/master/project-dog-classification
3. Hsu, David. "Using Convolutional Neural Networks to Classify Dog Breeds", Stanford University. http://cs231n.stanford.edu/reports/2015/pdfs/fcdh_FinalReport.pdf
4. Karen Simonyan, Andrew Zisserman "Very Deep Convolutional Networks for Large-Scale Image Recognition" Cornell University https://arxiv.org/abs/1409.1556
5. LFW Dataset, Udacity https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip
6. Dog Image Dataset, Udacity https://s3-us-west-1.amazonaws.com/udacity-aind/dogproject/dogImages.zip
7. https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html
8. https://medium.com/@ayyucekizrak/derine-daha-derine-evrişimli-sinir-ağları-2813a2c8b2a9
9. https://becominghuman.ai/image-data-pre-processing-for-neural-networks-498289068258
10. https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e