UDACITY
MACHINE LEARNING ENGINEERING NANODEGREE

Capstone Project Report

DOG BREED CLASSIFIER WITH PYTORCH

EROL ŞEN

July 11st 2021
GERMANY

1. DEFINITION

1.1 Project Overview

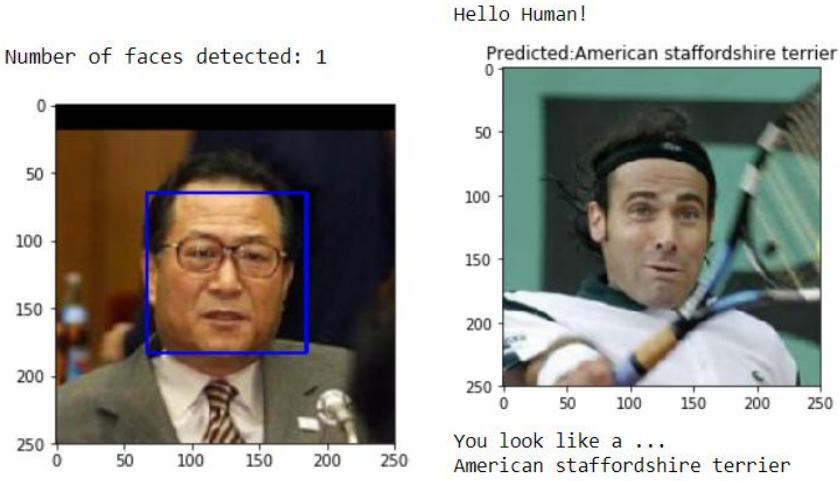
This project aims to identify dog breeds by using Convolutional Neural Networks with PyTorch which is developed by Facebook.

The basic idea is building a pipeline to process real-world, user-supply images. Given an image of a dog or a human, the algorithm will try to identify it.

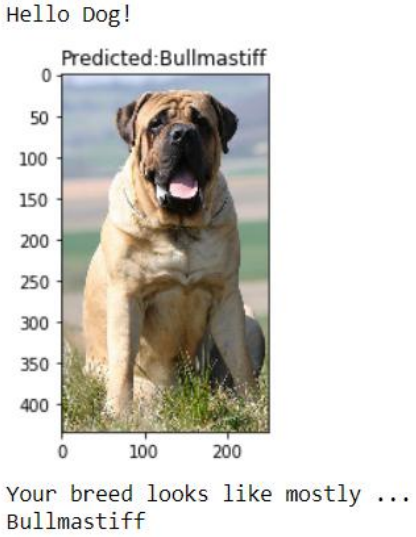
1.2 Problem Statement

In the project, a Dog Breed Classifier needs to be built by using Convolutional Neural Network. After the model which predicts Dog Breeds, user will be able to give an input image and the model will tell which dog breed it is.

If the given image has a human, face detector (that I created a face detector by using Haar Cascade) will try to detect human face, and in the same time, the model will try to detect any dog breed as well.



But if the given image has a dog, the model will identify which dog breed it is.



But the given image is out of one of these classes (human or dog breed), the model says that it can't identify and dog or human.

1.3 Metrics

The dataset has been split 3 parts, which are train, validation and test dataset. During training, the model will try to learn from the train dataset and it will predict itself in the validation dataset according to what it learned in the training dataset. After all training process, the model will try itself in the test dataset.

In order to calculate performance I am going to choose "**accuracy**". Accuracy is a rate of total images classified correctly by the model.

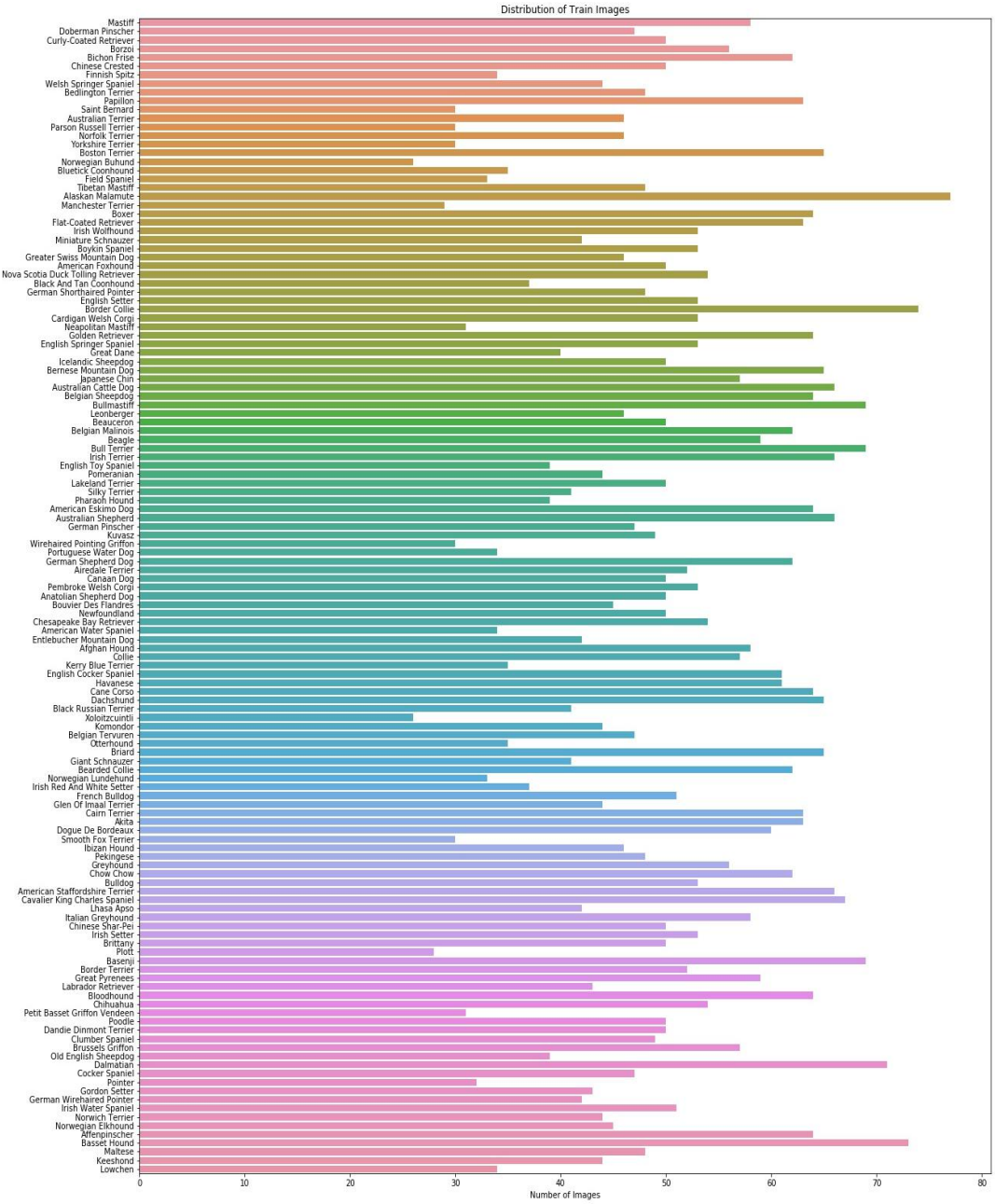
$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{All Samples}}$$

2. ANALYSIS

2.1 Data Exploration

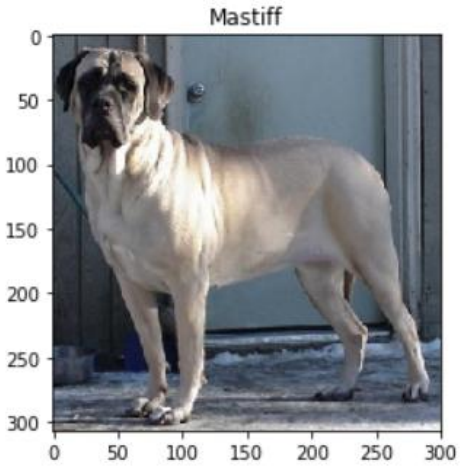
In the project environment, there are 13233 human images and 8351 dog images which are provided by Udacity.

In the schema below, distribution of classes can be saw.



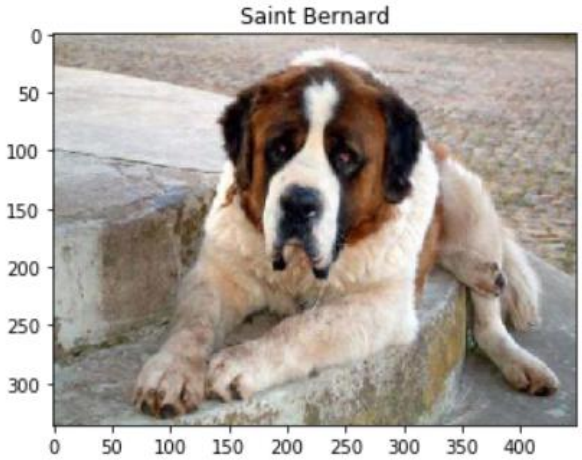
- Train dataset has 6680 images,

```
display_image(train, "Mastiff")
```



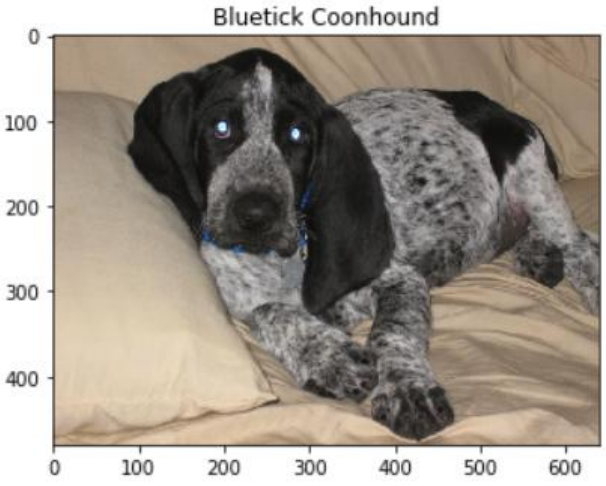
- Valid dataset has 835 images

```
display_image(test, "Saint Bernard")
```



- Test dataset has 836 images.

```
display_image(valid, "Bluetick Coonhound")
```



2.2 Algorithms and Techniques

As you may remember in “Problem Statement”, in order to detect faces, I will use Haar Cascade by using OpenCV.

On the other hand, in order to detect dog, I will use pre-trained VGG16 model, and I will also compare the accuracy of the pre-trained model.

But to detect dog breeds, I will use 2 different model which one of is from scratch, I will create my own CNN model, and other one is from Res-Net50 transfer learning with PyTorch.

Working with Res-Net50 transfer learning model, I will use all layers except prediction layer. So that, I will make it customize and I will be able to train the model in my dog breed dataset.

2.3 Benchmark

The scratch model that I will create must reach accuracy bigger than 10%, and other transfer learning model must reach accuracy bigger than 60% too. These are the limitations to pass this project.

3. Methodology

3.1 Data Preprocessing

To work with VGG16, all images must be 224x224, because of that, all images need to be resized as 224x224. For this process, PyTorch facilitates what we need to do.

```
import torchvision.transforms as transforms
```

By using "**transforms**" we can resize our images easily. I also want the images to be augmented, for that "**transforms**" helps us. Except resize, "**Horizontal Flip**", "**Random Rotation**" processes help the model much effective.
During training, these images come to as Flipped or rotated randomly and when the model learns, these steps make it difficult.

```
### Define Transformations
data_transform = transforms.Compose([transforms.RandomResizedCrop(224),
                                     transforms.RandomHorizontalFlip(p=0.5),
                                     transforms.RandomRotation(15),
                                     transforms.ToTensor(),
                                     transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                                          std=[0.229, 0.224, 0.225])])
```

3.2 Implementation

In this section, I will explain implementation of the algorithms which detect human faces, and dogs. I will also talk about VGG16 model, scratch CNN model and ResNet50 model.

- **Step 1:**
  First implementation was detecting human faces. To detect faces, I used "HaarCascades" which is one of the

pre-trained detectors by OpenCV. Implementing is really easy

```
# extract pre-trained face detector
face_cascade = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_alt.xml')

def face_detector(img_path):
    img = cv2.imread(img_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray)
    return len(faces) > 0
```

This function returns True or False according to the image.

- **Step 2:**

  Second implementation was dog detector. For that, VGG16 was used and also it was used for comparing the performance of VGG16 and the other models.

```
import torch
import torchvision.models as models

# define VGG16 model
VGG16 = models.vgg16(pretrained=True)

# check if CUDA is available
use_cuda = torch.cuda.is_available()

# move model to GPU if CUDA is available
if use_cuda:
    VGG16 = VGG16.cuda()
```

After this step, VGG16 model is ready to go.
In order to load and process the images, I defined a function called "**image_loader(img_path)**".

```
def VGG16_predict(img_path):

    img = image_loader(img_path)

    if use_cuda:
        img = img.cuda()

    output = VGG16(img)

    return torch.max(output,1)[1].item() # predicted class index
```

Prediction function is defined as well, this function is needed for dog detector function.

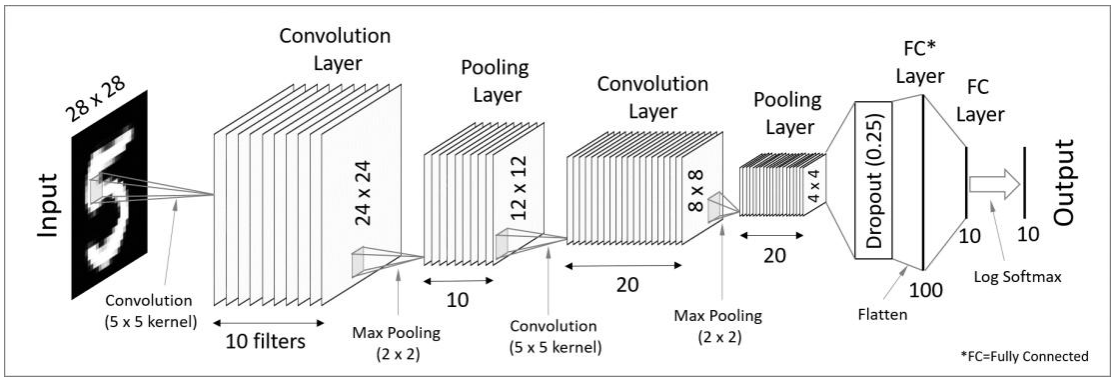
```
### returns "True" if a dog is detected in the image stored at img_path

def dog_detector(img_path):

   idx = VGG16_predict(img_path)

    if idx >= 151 and idx <= 268: return True
   else: return False

print(dog_detector(dog_files_short[13]))
print(dog_detector(human_files_short[13]))

>>> True
>>> False
```

After all these steps, I tested my dog detector in all datasets (human and dog)

```
detected_dog_in_humans = 0
detected_dog_in_dogs = 0
for human_path, dog_path in tqdm(zip(human_files_short, dog_files_shor
t), total=100):

    if dog_detector(human_path):
        detected_dog_in_humans += 1

    if dog_detector(dog_path):
        detected_dog_in_dogs += 1
        print(f"Percentage of detected dog faces in first 100 human file
s: {detected_dog_in_humans}")

print(f"Percentage of detected dog faces in first 100 dog files: {dete
cted_dog_in_dogs}")
>>> Percentage of detected dog faces in first 100 human files: 0
>>> Percentage of detected dog faces in first 100 dog files: 100
```

• **Step 3:**

As a step 3, I defined my Convolutional Neural Network model from scratch.



My Neural Network has 3 Convolutional layers which are connected by Pooling Layers, 1 Flatten Layer, 1 Dropout Layer and 2 Fully-Connected Layers which are connected by Dropout Layer too.

After 50 epochs, my CNN model obtains **16% test accuracy** which is really worst but it is enough for passing this section. I didn't dwelt on improving my CNN model because I will go with Transfer Learning model.

• **Step 4:**

In this last section, I will describe what I have done in Transfer Learning.

As a pre-trained model, I chose ResNet50 model. Before I have worked with this model and I have some idea what it does.

Pre-trained models have weights in their layers, that's why these models learn fast and effectively. What I did is, froze parameters and remove prediction layers (fully-connected layers). So that, the model is ready to train.

After 50 epochs later, the model obtains 71% test accuracy which it is not bad but not good. It is fair enough to pass this section.

3.3 Refinement

Before trained the CNN model, I got really low accuracy about 7% in test accuracy. I have decided to change number of epochs and optimizer. I changed the epoch as 50 and the optimizer as "**Adam**" and as learning rate, I determined 0.001. After these changes, I got 17% test accuracy, it is also really worst but fair enough to pass this section.

I also had same problem in transfer learning. Same steps were applied to the transfer learning model, and it got 71% test accuracy. I believe that if I increase the number of epochs, I got much better test accuracy.

Issues in Github:
https://github.com/ierolsen/Udacity-Dog-Breed-Classifier/issues/8
https://github.com/ierolsen/Udacity-Dog-Breed-Classifier/issues/9

4. Result

4.1 Model and Evaluation and Validation

The dataset has split 3 different parts which are train, validation and test. In training dataset, the model learns what images are and during training, tries to test itself in the validation dataset. End of the training, model tests itself in the test dataset.

```
Epoch: 1        Training Loss: 2.874242         Validation Loss: 1.626871
Validation loss decreased (inf --> 1.626871). Saving the model
Epoch: 1 took 81.176 seconds.

Epoch: 2        Training Loss: 1.440387         Validation Loss: 1.285117
Validation loss decreased (1.626871 --> 1.285117). Saving the model
Epoch: 2 took 79.340 seconds.

Epoch: 3        Training Loss: 1.259456         Validation Loss: 1.276030
Validation loss decreased (1.285117 --> 1.276030). Saving the model
Epoch: 3 took 79.137 seconds.

Epoch: 4        Training Loss: 1.148424         Validation Loss: 1.229575
Validation loss decreased (1.276030 --> 1.229575). Saving the model
Epoch: 4 took 79.073 seconds.

Epoch: 5        Training Loss: 1.079556         Validation Loss: 1.129032
Validation loss decreased (1.229575 --> 1.129032). Saving the model
Epoch: 5 took 78.959 seconds.
```

As I mentioned in "**Refinement**" section, the model couldn't learn well in first version of the notebook. Then I made some changes in layers and optimizer.

First, I used "**SGD**" as a optimizer and I set 0.05 learning rate, but I didn't work well. Then I decided to decrease learning rate and I determined it 0.001 and I set an optimizer called "**Adam**".

This changed affected the models (CNN and Transfer Learning models) in great way. Test accuracies increased for 2 models.

4.2 Justification

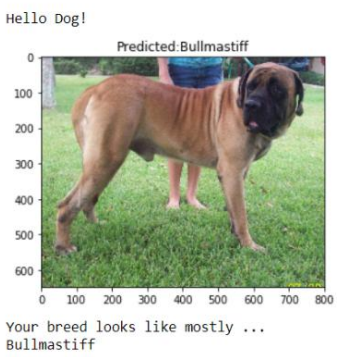
The model's performance is like what I excepted. If I increase the number of epochs, probably I can obtain much better result. Even 71% accuracy is really enough to predict identification of dog breeds.

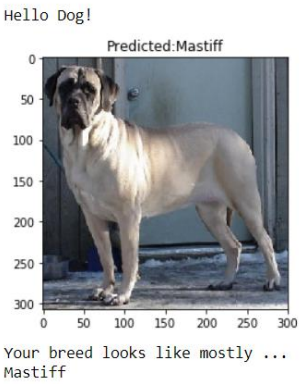
	Number of Epochs	**Test Loss**	**Test Accuracy**
CNN from Scratch	50	3.558069	16% (136/836)
ResNet 50 Transfer Learning	50	1.236936	71% (596/836)

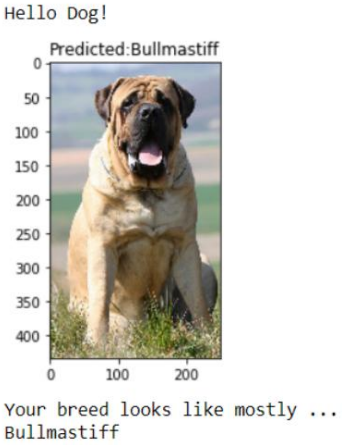
5. Conclusion

5.1 Free-Form Visualization

I would like to share some predictions which are made by the Transfer Learning model.

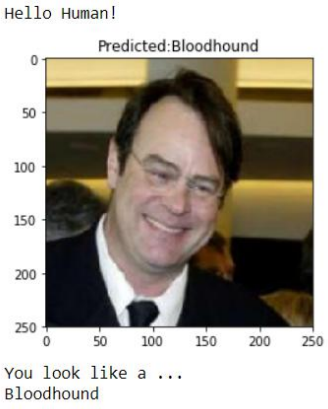


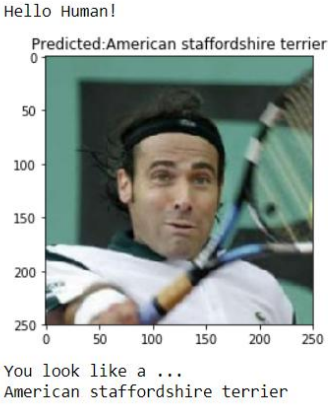


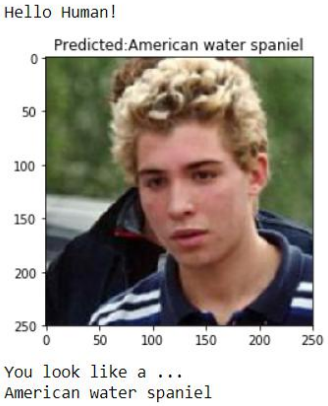


If given image of a human, the face detector algorithm predicts it as human and also dog predictor function called "**predict_breed_transfer"** tries to predict it as a dog breed.

For instance:







5.2 Reflection

Summary of my steps:

- **Step 0:** Import Libraries and Datasets
- **Step 1:** Detect Humans by using Haar Cascade provided by OpenCV
- **Step 2:** Detect Dogs by using VGG16 pre-trained model
- **Step 3:** Create a CNN Model from scratch
- **Step 4:** Create a Transfer Learning model (ResNet50)
- **Step 5:** Write a Main Function using the ResNet 50 Model and the Human Face Detector Function.
- **Step 6:** Test the Algorithm if it works

The section I found difficult was, creating CNN model from scratch, because before I didn't work with PyTorch and I had no more knowledge about it. And also completing transfer learning prediction function called "**predict_breed_transfer**" and training algorithm forced me because of the same reason.

But after this challenging project, I earn self-confident in PyTorch and Machine Learning.

5.3 Improvement

To improve the model, it can be added more dog breed classes. And also, transfer learning model can be changed, instead of ResNet50, it can be chosen "**Inception**", this changes may provide high accuracy.

References

1. ***Image Classification with PyTorch***
https://www.youtube.com/watch?v=zFA8Cm13Xmk
2. ***PyTorch Pre-Trained Models***
https://pytorch.org/docs/0.3.0/torchvision/models.html
3. ***PyTorch Transfer Learning Tutorial***
https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html
4. ***PyTorch ResNet Tutorial***
https://pytorch.org/hub/pytorch_vision_resnet/
5. ***Original Udacity Project***
https://github.com/udacity/deep-learning-v2-pytorch/tree/master/project-dog-classification
6. ***Udacity Transfer Learning GitHub Project***
https://github.com/udacity/deep-learning-v2-pytorch/blob/master/intro-to-pytorch/Part%208%20-%20Transfer%20Learning%20(Solution).ipynb