Machine Learning Engineer Nanodegree

Capstone Project

Dog Breed Classification

Gojko Galonja

April 28th 2020.

Definition

Project Overview

Humans live through their eyes, they feel, react and act based on what they see in most of the cases. Cognitive observations are the primary shaper of one's life. Computer vision is a discipline of deep learning that has a goal to mimic visual skills that humans have, such as detecting shapes, colors, recognizing faces and objects, and much more.

In this project, we will solve a problem of dog breed detection, by simply making the user input an image of a dog he/she wants to detect and our application will return the prediction of a breed of that dog. The application has a funny side of it as well, and that is if the human face is detected, it will state that and output the dog breed that face is mostly similar to.

The model will be trained with transfer learning technique, with a different model that has been previously developed and trained upon thousands of dog images (so called ImageNet database).

This will help us train our model faster and make it more precise and robust.

Problem Statement

In this project, Deep learning, more specifically Convolutional Neural Networks technology will be used in order to finish and execute the given problem.
Using larger quantities of images as our main source of data, we will create an image classification algorithm that will give us certain output.

This project is quite known in the Machine learning world. It’s been used and practiced in universities, bootcamps and courses, and with a good reason.

In this project, Deep learning, more specifically Convolutional Neural Networks technology will be used in order to finish and execute the given problem.
Using larger quantities of images as our main source of data, we will create an image classification algorithm that will give us certain output.

This project is quite known in the Machine learning world. It's been used and practiced in universities, bootcamps and courses, and with a good reason.
The goal is to build a web or mobile application that will detect whether a dog or a human face is detected in a given image.

The dataset used for training consists of two classes - dogs and humans.
We are building this application in order to detect if a dog is on the image, and if so, which breed is it. If the human face is detected, then try to figure out which dog breed the human face is similar to. Human face detection will be done with OpenCV technology, more specifically with its builtin Haar Cascade algorithm.
The reason behind the question why we are building this is simply to detect dog breed and on the human face part, to bring some fun in it.

In order to build this algorithm, **supervised learning** methods will be applied.
Below are the examples of both outputs of the detection and classification.

Metrics

For metrics, classification accuracy approach will be used, as we have supervised learning algorithms.

Simple formula for calculating accuracy is:

correct predictions / total number of classifications = accuracy percentage

Classification accuracy is a good measure when there are a similar number of data samples per class, in contrast, in case if the dataset is quite imbalanced in that sense, then it wouldn't be a good choice for metrics.

Analysis

Data Exploration & Visualizations

The dataset consists of two classes - human and dog images.
The number of human images is 13233 and the number of dog images is 8351.

The testing images are real-world, user-like images that mimics the real input of the users of the application.

The dataset is already contained in the Udacity data folders.

1. Dog images folder - **dog_image**

    - Dog image folder consists of train, valid and test folder with each containing 133 dog breed folders with it's own pictures in it
    - Number of dog pictures in general is 8351, so roughly the number of images across the dog breed folders can be 62.78 which represents mean value
    - Distribution between Train,Test and Validation sets is:
        - Train - 6680 images, which makes 80%
        - Test - 836 images, which makes 10%
        - Validation - 835 images, which makes 10%

2 Human images folder - **/lfw**

    - Human image folder consists of names of people containing their image (one per each person/folder)
    - Number of images in human folder in total is 13233

We have a presence of similarities between certain dog breeds. Even humans would struggle to determine which dog breed resembles each dog in the images showcased below.

Also, we have dogs that are of the same breed, but look different. For example these three labradors:

All three of these dogs are labradors, but the primary differences between them is the color.

Our model is trying to conquer these high intra-class variations and classify all of these dogs as the same breed.

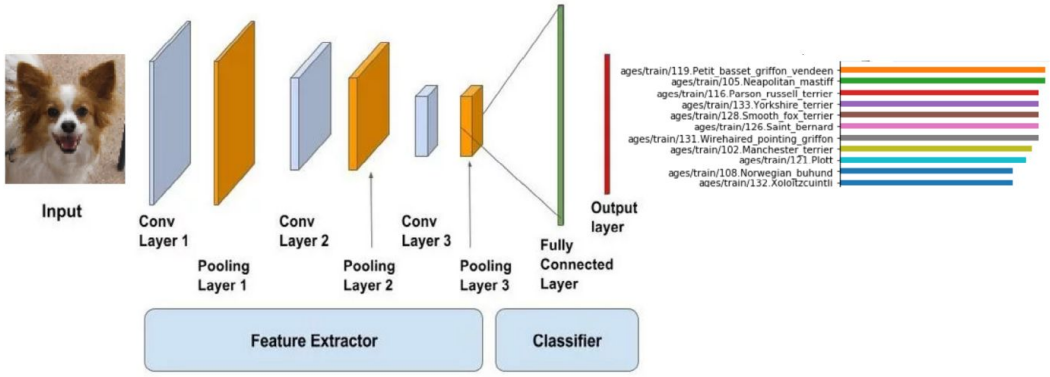
Algorithms, Techniques, Design and Benchmarking

With a clear intuition, this problem will be solved using deep learning.
Key of solving this problem will be in applying CNN architecture in order to make a model that will do the detection/classification.
CNN is short of Conventional Neural Networks which is a technique used specifically for solving classification, detection, prediction computer vision problems.

First, the model will be created from scratch. Then another model will be created using transfer learning technique to create another model.

The reason we use transfer learning is because the dog dataset is insufficient in order to create a model that will give good dog breed prediction.
VGG-16 model is a state-of-the-art model that is trained on an exponentially larger dataset.

Visual representation of CNN network applied for dog breed classification (*** *Image taken from Google images search* ***):



Approach to designing this project's architecture

Code snippet for preprocessing and loading data::

```
transformer = transforms.Compose([
                        transforms.Resize((224, 224)),
                        transforms.ToTensor(),
                        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])])

train_transformer = transforms.Compose([
                        transforms.Resize(224),
                        transforms.RandomResizedCrop(224),
                        transforms.ToTensor(),
                        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])])

train_data = datasets.ImageFolder(root='/data/dog_images/train/', transform=train_transformer)
test_data = datasets.ImageFolder(root='/data/dog_images/test/', transform=transformer)
valid_data = datasets.ImageFolder(root='/data/dog_images/valid/', transform=transformer)

train_loader = torch.utils.data.DataLoader(train_data,
                                           batch_size=8,
                                           shuffle=True,
                                           num_workers=2)
test_loader = torch.utils.data.DataLoader(test_data,
                                           batch_size=8,
                                           shuffle=True,
                                           num_workers=2)
valid_loader = torch.utils.data.DataLoader(valid_data,
                                            batch_size=8,
                                            shuffle=True,
                                            num_workers=2)

loaders_scratch = {'train': train_loader,
                   'valid': valid_loader,
                   'test': test_loader}
```

Architecture will be broken down into the following steps:

- Create human face detector / Detect humans - Use OpenCV Haar Cascade which returns positive if human face is detected and negative on the opposite
- Create dog detector - Use VGG-16 (pretrained model) in order to detect the dogs
- Write three separate data loaders for the training, validation, and test datasets of dog images (located at dog_images/train, dog_images/valid, and dog_images/test)

- Create a model from the scratch
    - Create a CNN model architecture to classify dog breed
    - Specify Loss Function and Optimizer (CrossEntropyLoss and Stochastic Gradient Descent - SGD optimizer)
    - Train, Validate and Test the Model
    - Architecture for scratch model:

      Each Convolutional layer is followed by a MaxPooling layer, and at the last combination of Convolutional and MaxPooling layer, flattening is performed.

      Conv1 -> MaxPool1 -> Conv2 -> MaxPool2 -> Conv3 -> MaxPool3 -> Conv4 -> MaxPool4 -> Conv5 -> MaxPool5 -> Flattening followed by two hidden layers in order to get the probability prediction.

      #1 Layer - Conv Layer: Tensor dimensions - 224x224x3 converting to 222x222x16 | MaxPool Layer: Conversion dimensions - 111x111x16

      #2 Layer - Conv Layer: Tensor dimensions - 111x111x16 converting to 109x109x32 | MaxPool Layer: Conversion dimensions - 54x54x32

      #3 Layer - Conv Layer: Tensor dimensions - 54x54x32 converting to 52x52x64 | MaxPool Layer: Conversion dimensions - 26x26x64

      #4 Layer - Conv Layer: Tensor dimensions - 26x26x64 converting to 24x24x128 | MaxPool Layer: Conversion dimensions - 12x12x128

      #5 Layer - Conv Layer: Tensor dimensions - 12x12x128 converting to 10x10x256 | MaxPool Layer: Conversion output dimensions to 5x5x256

- Training image sample:

```
Epoch: 1        Training Loss: 0.000732         Validation Loss: 0.005819
Validation loss: 0.005819       Minimal Validation Loss: inf
Epoch: 2        Training Loss: 0.000729         Validation Loss: 0.005773
Validation loss: 0.005773       Minimal Validation Loss: 0.005819
Epoch: 3        Training Loss: 0.000725         Validation Loss: 0.005648
Validation loss: 0.005648       Minimal Validation Loss: 0.005773
Epoch: 4        Training Loss: 0.000714         Validation Loss: 0.005461
Validation loss: 0.005461       Minimal Validation Loss: 0.005648
Epoch: 5        Training Loss: 0.000708         Validation Loss: 0.005438
Validation loss: 0.005438       Minimal Validation Loss: 0.005461
Epoch: 6        Training Loss: 0.000702         Validation Loss: 0.004838
Validation loss: 0.004838       Minimal Validation Loss: 0.005438
Epoch: 7        Training Loss: 0.000689         Validation Loss: 0.005263
Epoch: 8        Training Loss: 0.000679         Validation Loss: 0.005654
Epoch: 9        Training Loss: 0.000673         Validation Loss: 0.005472
Epoch: 10       Training Loss: 0.000665         Validation Loss: 0.005536
Epoch: 11       Training Loss: 0.000659         Validation Loss: 0.004953
Epoch: 12       Training Loss: 0.000654         Validation Loss: 0.005575
Epoch: 13       Training Loss: 0.000646         Validation Loss: 0.005046
Epoch: 14       Training Loss: 0.000641         Validation Loss: 0.005122
Epoch: 15       Training Loss: 0.000634         Validation Loss: 0.005247
Epoch: 16       Training Loss: 0.000625         Validation Loss: 0.005460
Epoch: 17       Training Loss: 0.000620         Validation Loss: 0.005563
Epoch: 18       Training Loss: 0.000613         Validation Loss: 0.004057
Validation loss: 0.004057       Minimal Validation Loss: 0.004838
Epoch: 19       Training Loss: 0.000605         Validation Loss: 0.004686
Epoch: 20       Training Loss: 0.000598         Validation Loss: 0.004769
Epoch: 21       Training Loss: 0.000593         Validation Loss: 0.005298
Epoch: 22       Training Loss: 0.000582         Validation Loss: 0.005070
Epoch: 23       Training Loss: 0.000576         Validation Loss: 0.004144
Epoch: 24       Training Loss: 0.000564         Validation Loss: 0.004554
Epoch: 25       Training Loss: 0.000558         Validation Loss: 0.005358
Epoch: 26       Training Loss: 0.000546         Validation Loss: 0.003863
Validation loss: 0.003863       Minimal Validation Loss: 0.004057
Epoch: 27       Training Loss: 0.000539         Validation Loss: 0.003949
Epoch: 28       Training Loss: 0.000528         Validation Loss: 0.004268
Epoch: 29       Training Loss: 0.000514         Validation Loss: 0.004546
Epoch: 30       Training Loss: 0.000511         Validation Loss: 0.004890
```

- Testing image sample:

```
Test Loss: 3.742801


Test Accuracy: 15% (126/836)
```

- Create a model with **Transfer Learning** technique (With VGG-16 pretrained model) to train new model
    - Create a CNN model architecture to classify dog breed
    - Specify Loss Function and Optimizer (CrossEntropyLoss and Adam optimizer)
    - Train, Validate and Test the Model
    - Architecture for transfer learning mode:

      Even though VGG16 is a big network and might be slow to train, but it's a great building block, simple and relatively easy to implement.

      VGG16 uses 3x3 convolution and 2x2 pooling throughout the whole network.

      Due to its size, it shows that deeper neural networks usually give better results.

      Since the VGG16 has dog detection in the model itself, it shares a lot of similarities with dog breed detection, so I kept most of the pre-trained parameters, but changed the last layer to match with the total number of project's out-features.

    - Last layer architecture:

```
(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)
)
```

- Image for training results:

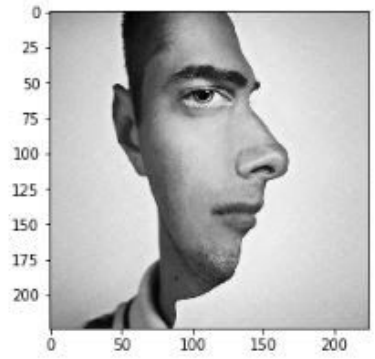
```
Epoch: 1        Training Loss: 0.000336         Validation Loss: 0.002116
Validation loss: 0.002116       Minimal Validation Loss: inf
Epoch: 2        Training Loss: 0.000204         Validation Loss: 0.001605
Validation loss: 0.001605       Minimal Validation Loss: 0.002116
Epoch: 3        Training Loss: 0.000144         Validation Loss: 0.001261
Validation loss: 0.001261       Minimal Validation Loss: 0.001605
Epoch: 4        Training Loss: 0.000116         Validation Loss: 0.000450
Validation loss: 0.000450       Minimal Validation Loss: 0.001261
Epoch: 5        Training Loss: 0.000080         Validation Loss: 0.000514
Epoch: 6        Training Loss: 0.000069         Validation Loss: 0.001168
Epoch: 7        Training Loss: 0.000056         Validation Loss: 0.001570
Epoch: 8        Training Loss: 0.000053         Validation Loss: 0.000244
Validation loss: 0.000244       Minimal Validation Loss: 0.000450
Epoch: 9        Training Loss: 0.000046         Validation Loss: 0.000804
Epoch: 10       Training Loss: 0.000045         Validation Loss: 0.000147
Validation loss: 0.000147       Minimal Validation Loss: 0.000244
Epoch: 11       Training Loss: 0.000033         Validation Loss: 0.000413
Epoch: 12       Training Loss: 0.000037         Validation Loss: 0.000373
Epoch: 13       Training Loss: 0.000028         Validation Loss: 0.000365
Epoch: 14       Training Loss: 0.000032         Validation Loss: 0.000411
Epoch: 15       Training Loss: 0.000030         Validation Loss: 0.000330
Epoch: 16       Training Loss: 0.000030         Validation Loss: 0.001104
Epoch: 17       Training Loss: 0.000029         Validation Loss: 0.000216
Epoch: 18       Training Loss: 0.000023         Validation Loss: 0.000094
Validation loss: 0.000094       Minimal Validation Loss: 0.000147
Epoch: 19       Training Loss: 0.000019         Validation Loss: 0.000316
Epoch: 20       Training Loss: 0.000025         Validation Loss: 0.000217
Epoch: 21       Training Loss: 0.000019         Validation Loss: 0.000948
Epoch: 22       Training Loss: 0.000025         Validation Loss: 0.000014
Validation loss: 0.000014       Minimal Validation Loss: 0.000094
Epoch: 23       Training Loss: 0.000012         Validation Loss: 0.000293
Epoch: 24       Training Loss: 0.000015         Validation Loss: 0.000327
Epoch: 25       Training Loss: 0.000024         Validation Loss: 0.000375
Epoch: 26       Training Loss: 0.000019         Validation Loss: 0.000068
Epoch: 27       Training Loss: 0.000016         Validation Loss: 0.000768
Epoch: 28       Training Loss: 0.000016         Validation Loss: 0.000010
Validation loss: 0.000010       Minimal Validation Loss: 0.000014
Epoch: 29       Training Loss: 0.000013         Validation Loss: 0.000097
Epoch: 30       Training Loss: 0.000014         Validation Loss: 0.000203
```

- Predict the dog breed / Test the model **accuracy**
  - Test sample image:

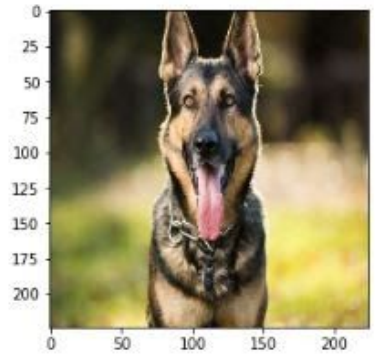
```
Test Loss: 2.105189


Test Accuracy: 62% (520/836)
```

- Write an algorithm that will:
    - if a **dog** is detected in the image, return the predicted breed
    - if a **human** is detected in the image, return the resembling dog breed
    - if **neither** is detected in the image, provide output that indicates an error
- **Test** algorithm on sample images
    - Examples of testing on sample images:

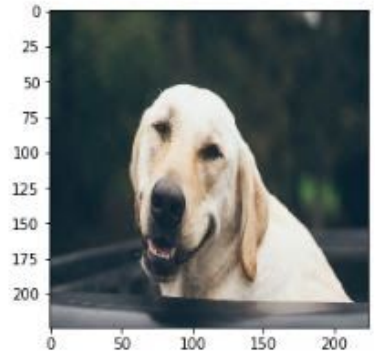
```
No humans nor dogs detected, please input another image.
```



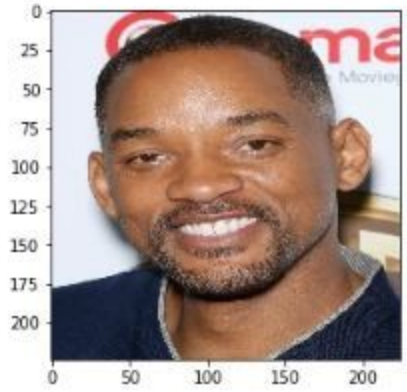

```
Dog detected and it's breed is German pinscher.
```



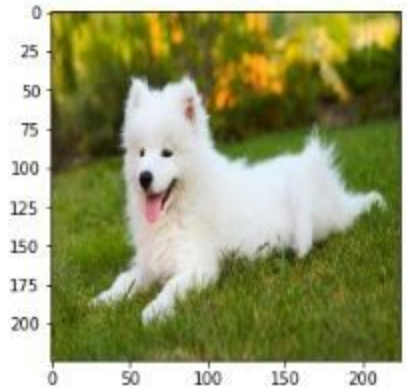

```
Dog detected and it's breed is Labrador retriever.
```



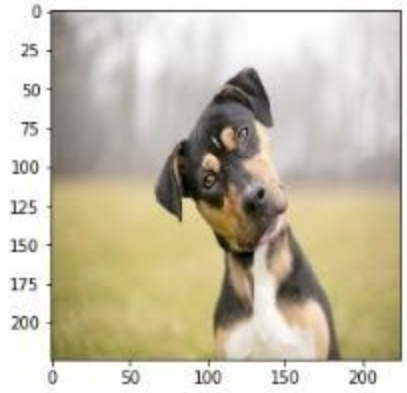
Human detected and it looks like Dogue de bordeaux



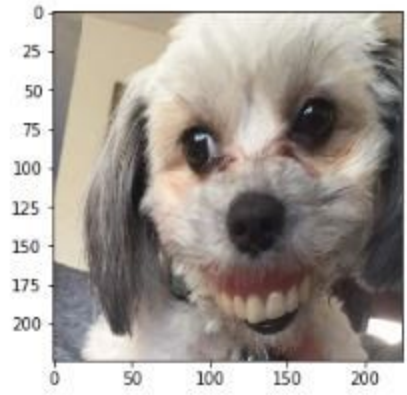
Dog detected and it's breed is American eskimo dog.



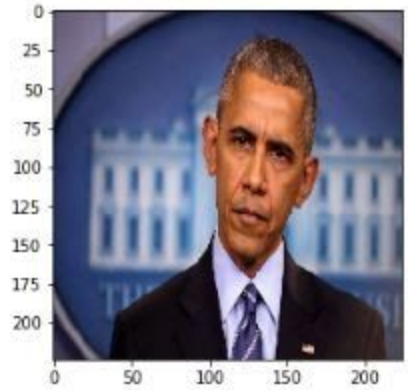
Dog detected and it's breed is Greater swiss mountain dog.



```
Dog detected and it's breed is Pekingese.
```




```
Human detected and it looks like German pinscher
```



Summary

Reflecting and showcasing potential improvements

Summarization of a process of executing this project:

1. Definition of a project has been stated and the necessary dataset has been acquired
2. Dataset was thoroughly analyzed, explored, preprocessed and loaded
3. Model from scratch was created/trained and evaluated
4. Using transfer learning technique, another model was developed. Changed its final layer and adjusted it to work corresponding to the number of classes in the dataset.
5. Transfer Learning Model was created/trained.
6. Transfer Learning Model was tested and evaluated.
7. All accuracy requirements were met.

Possible improvements:

1. More epochs could be introduced in the training to see at which point the model could go without overfitting.
2. Apart from the last layer, all other layers are frozen within the VGG16 pretrained model. More of these layers could be "unfrozen" and some tweaking and fine-tuning could be performed on them.
3. VGG16 pretrained model was used, which is maybe not the best option out there, so with some research another pretrained model could be studied, introduced and trained upon (Model with ResNet architecture).