Machine Learning Engineer Nanodegree

Capstone Project

Abdulrahim Alotaibi
May 15th, 2020

I. Definition

Project Overview

Machine learning, a well-known subset of Artificial Intelligence, is widely used to make predictions and decision that help achieving goals without being explicitly programmed to do. Deep learning is known a one of the famous members of machine learning family. Deep learning uses neural network to make decision that affect real life. Deep learning applications include computer vision, speech recognition and social networks filtering.

Dog, also known as man's best friend, have a strong bond with humans in a long bond that traced back at least 15,000 years ago. Dogs have a various number of breeds. One of the difficult tasks are identifying a dog's breed due to large number of breeds. The challenge in this project is using deep learning Convolutional Neural Network to perform image classification to identify a dog breed. We used two types of Convolutional Neural Network, the first was made from scratch and the second was made using pretrained models such as VGG16. The results were interesting, and some point of improving has been noticed and presented in this report,

Problem Statement

The objective of this project is designing an algorithm that could be used in either a mobile application of a website that will run by taking a user input data, in this case it is going to an image, and use deep learning Convolutional Neural Network to identify what is found in the picture. If a dog is found in the picture, the algorithm will display the estimated dog's breed. In another case, if human face found in the picture, the algorithm will display the most dog breed resembling. The solution will be using a pre-trained VGG16 model with accuracy that is equal to 80 percent.

Metrics

When it comes to metrics, a very well-known metric will be used, and it called Accuracy which represent the percentage of the correct predictions that has been done on the test data. In, the creation of the Convolutional Neural Network from scratch to classify dog breeds, we attempted to attain a test accuracy of at least 10%. In the step of the use of transfer learning to create a CNN that can identify dog breed from images, we attempted to attain at least 60% accuracy on the test set.

II. Analysis

Data Exploration and Visualization

The datasets that has been used in this project are strongly related to the problem. Two dataset that contains dogs and human images. The human dataset contains 13233 human images and the dog data set consist of 8351 dog images referenced with the dog breed that will be used in this project to classify them into 133 dog breeds. The images will be distributed into three categories include training, validating, and testing. All the images are different in sizes and shapes. As a result, the images will be normalized to be center cropped in working with the preprocessing in designing both CNN from scratch and CNN using transfer learning to make images with a size equal to 224 by 224 pixels. Since the data used in this project are pictures type here are some visualization of the pictures used to solve the problem.

Algorithms and Techniques

In order to solve this problem, we will use one of the useful deep learning algorithms that is specialized in computer vision and it is Convolutional Neural Networks.

- Convolutional Neural Networks:
  It benefits of the very fact that the input consists of pictures and that they constrain the design in an exceedingly a lot of smart method. CNN applications can be almost everywhere from computer vision to sentiment analysis.
- Convolutional Neural Networks architecture:
  The architecture of CNN is containing Convolutional Layer, Pooling Layer, and Fully Connected Layer.

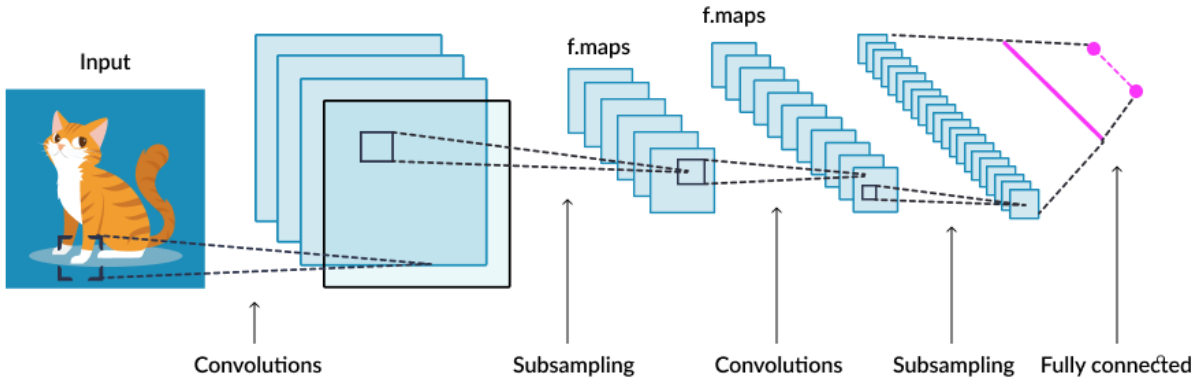


Figure 1 representation of Convolutional Neural Network architecture.

In this project, a creation of two Convolutional Neural Network has been done. One of them made from scratch, the second has been made using transfer learning.

Benchmark

In Benchmark, Accuracy has been chosen for Convolutional Neural Network from scratch and the goal was that to attain a test accuracy of at least 10% and what happened is the we got 11% in the accuracy. And for Convolutional Neural Network using transfer learning we tried to attain at least 60% accuracy on the test set. Here we got 80% accuracy when tested.

III. Methodology

Data Preprocessing

In the data preprocessing, we used the same data preprocessing for the CNN made form scratch and the CNN using transfer learning. Here is the code for the data preprocessing:

```
import os
from torchvision import datasets
from torch.utils.data import DataLoader


### TODO: Write data loaders for training, validation, and test sets
## Specify appropriate transforms, and batch_sizes

dog_data_dir = '/data/dog_images/'
train_dir = os.path.join(dog_data_dir , 'train/')
valid_dir = os.path.join(dog_data_dir , 'valid/')
test_dir = os.path.join(dog_data_dir , 'test/')

normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])
transformer = transforms.Compose([transforms.Resize(256),   # resize the image.
                           transforms.CenterCrop(224),    # Crop the center of the image.
                           transforms.ToTensor(),    # convert the image into PyTorch tensor.
                           normalize ]   # normalize the image
                         )

train_data = datasets.ImageFolder( train_dir , # directory path
                                   transform = transformer  # transform the image
                                  )
valid_data = datasets.ImageFolder( valid_dir , transform = transformer)
test_data = datasets.ImageFolder( test_dir , transform = transformer)

# load images
train_data_loader = DataLoader(train_data, batch_size=20, shuffle=True, num_workers=0)
valid_data_loader = DataLoader(valid_data, batch_size=20, shuffle=True, num_workers=0)
test_data_loader = DataLoader(test_data, batch_size=20, shuffle=True, num_workers=0)
```

Implementation

- First, we need to make a human face detector and that can be done using OpenCV's implementation of Haar feature-based cascade classifiers.
- Then, we need to make a dog detector using VGG16 model, along with weights that have been trained on ImageNet, a very large dataset used for image classification and other vision tasks.
- After that, we need to create the Convolution Neural network from scratch. Here we will use Convolution Neural network layers such as Convolutional Layer, Pooling Layer, and Fully Connected Layer.
- The CNN from scratch architecture and the steps that has been done are as follow: first, in this CNN we need to create a three two-dimensional convolutional layers (Conv2d). the parameters of the first conv2d layer are 3 in_channels (number of input channels), 32 out_channels (number of produced channels), 3 kernel_size (basically the size of the filters) , stride of 2 means processing every other pixel and padding =1 if an additional layer that is added to the both sides of the input.
- After finishing the 3 conv2d layers. We need to realize that Convolutional Neural Network is linear function, so we need to add some nonlinearity hence we use relu. Also, pooling is another non-linear layer added and its job is downsampling.
- FC stands for Fully Connected and this type of layers tend to be in the end and its job is to take the previous layers' results and start classifying. Since the images are of dogs and we want to classify dog pictures into 133 dog breeds, the number of classes is going to be 133.

```python
import torch.nn as nn
import torch.nn.functional as F

# 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, 32, 3, stride= 2, padding= 1) # 1st convolutional layer
        self.conv2 = nn.Conv2d(32, 64, 3, stride= 2, padding= 1) # 2nd convolutional layer
        self.conv3 = nn.Conv2d(64, 128, 3, padding= 1) # 3rd convolutional layer
        self.pool = nn.MaxPool2d(2,2) # pooling to a mat of 2x2
        self.fc1 = nn.Linear(7*7*128, 500) # 1st Fully Connected layer
        self.fc2 = nn.Linear(500, 133) # 2nd Fully Connected layer
        self.dropout = nn.Dropout(0.30) # Dropout to help avoiding Over fitting

    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 = x.view(-1, 7*7*128)
        x = self.dropout(x)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

- We now move on to creating the Convolutional Neural Network using transfer learning.
- Since building a Convolutional Neural Network from scratch has a problem with accuracy. Here we tried to build a CNN using transfer learning. The idea and logic of using transfer learning is using a pretrained model on a large dataset ing this case the pretrained model is VGG16.
- Here we have used the following steps, we freeze the last convolutional layer of the network and only train the last fully connected (FC) layer which make a prediction of the 133 breeds. I hope it make sense that it is easier and also less time consuming.

```
import torchvision.models as models
import torch.nn as nn

## TODO: Specify model architecture
model_transfer = models.vgg16(pretrained= True) # get the vgg16 model

pre_trained = model_transfer.features.parameters()
for p in pre_trained :     # we stop the pretrained parameters
    p.requires_grad = False


# get the input size and change the classefier to 133 breeds
input_size = model_transfer.classifier[6]
input_size = input_size.in_features
model_transfer.classifier[6] = nn.Linear(input_size, 133)

# check for gpu
if use_cuda:
    model_transfer = model_transfer.cuda()
```

- After that we will move on to writing the algorithm that will takes a file path of an image and determines whether the image contains a human, dog, or neither. Also, it will return the predicted breed if a dog is detected in the image. Moreover, it will return the resembling dog breed if a human is detected in the image. Other than that, it will provide output that indicates an error if neither a dog nor human is detected in the image. Here is the code.

```
### 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

    # open image :
    image = Image.open(img_path)
    # create an image and present it
    plt.imshow(image)
    plt.show()
    # if statement for the app process
    if face_detector(img_path) is True :
        result = predict_breed_transfer( model_transfer, class_names, img_path)
        print(" WELCOME TO DOG APP! \n Human!, here is your dog breed: \n ", result)
    elif dog_detector(img_path) is True :
        result = predict_breed_transfer( model_transfer, class_names, img_path)
        print(" WELCOME TO DOG APP! \n Dog!, here is your breed: \n ", result)

    else :
        print("Welcome to dog app! \n Oops! \n Couldn't found a dog or human!")
```

Refinement

The major improvement that really has affected the model is the hyperparameter tuning. What happened is that I tried 5 epochs and the test gave me an accuracy that is less than 10% in the Convolutional Neural Networks that made from scratch but when I increased the number of epochs to 10 the accuracy increase to 11%.

IV. Results

Model Evaluation and Validation

As stated above that the dataset will be distributed into three categories, training, validation, and testing. Also, since we need to choose weather using the CNN made from scratch or the CNN using transfer learning. Here is accuracy test for CNN made from scratch:

```
# call test function
test(loaders_scratch, model_scratch, criterion_scratch, use_cuda)
```

```
Test Loss: 3497.604280


Test Accuracy: 11% (100/836)
```

Here is the accuracy test for CNN using transfer learning:

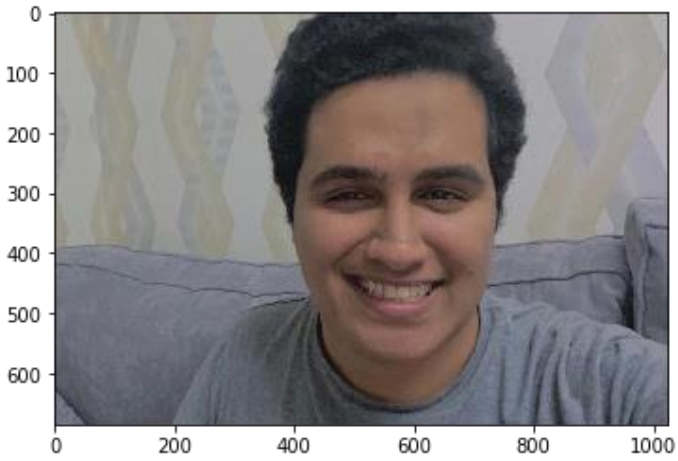
```
test(loaders_transfer, model_transfer, criterion_transfer, use_cuda)
```

```
Test Loss: 660.613758


Test Accuracy: 80% (673/836)
```

Based on that we made the decision of using CNN using transfer learning.

Here are some tests of using the algorithm:

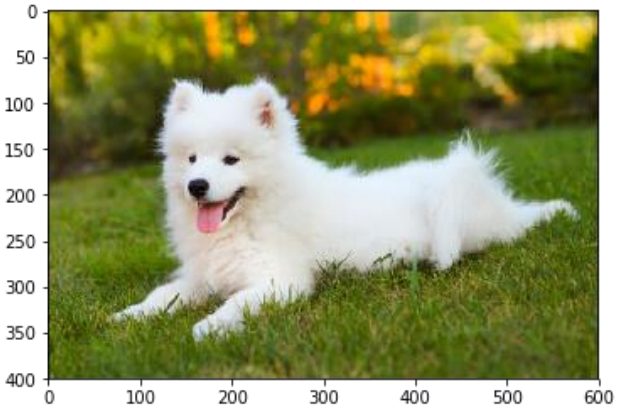



```
WELCOME TO DOG APP!
Human!, here is your dog breed:
 Dachshund
```


```
WELCOME TO DOG APP!
Human!, here is your dog breed:
 Poodle
```


```
WELCOME TO DOG APP!
Human!, here is your dog breed:
```

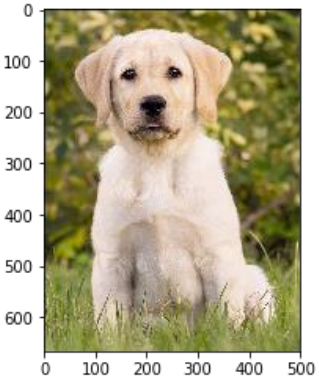
Chinese crested



WELCOME TO DOG APP!

Dog!, here is your breed:

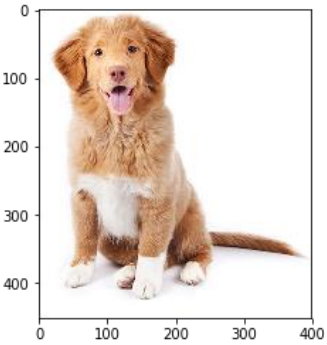
American eskimo dog



WELCOME TO DOG APP!

Human!, here is your dog breed:

Labrador retriever



WELCOME TO DOG APP!

Dog!, here is your breed:

Nova scotia duck tolling retriever

Justification

The results are more than what stated on the Benchmark. The accuracy of the CNN made from scratch is 11% which is higher than 10%. For that CNN made using transfer learning, the accuracy is 80% which is much higher than 60%.

V. Conclusion

Free-Form Visualization

When we were in the phase of test the algorithm, I tried to use a cat image and the result was that the algorithm detected it as a dog and I think that the reason behind that is due to the accuracy is 80% and the type of cats I chose was quite a bit close from a dog breed.

Reflection

This project has proved that whenever we want to make a Convolutional Neural Network from scratch, we should use a quite large dataset. Other than that, the accuracy will be very low and hence the problem will not be solved. Here come the benefits of using a pretrained models, in this project we used a VGG16 pretrained model that helped a lot in solving the problem.

Improvement

I think there is a big room for improvement in implementing this kind of CNN projects and here are some points:

- Increasing the num of images to help the model become better and better.

- Also, increasing the number of models and adding mode FC layeers will help improving the model.

- Finally, playing and changing with the hyperparameters will result in increasing the accuracy since it now just 80%.