Capstone Project Jose Rubio

Machine Learning Engineer Nanodegree November 25th, 2020

Dog Breed Classifier using CNN:

Project Overview

The idea is identifying the breed of a dog, inside of an image given as input, if a human is inside the picture, we have to identify the resembling dog breed by building a pipeline that can process real world user supplied images and identify an estimate of the canine's breed. This is a multi-class classification problem where we can use un supervised or unsupervised machine learning to solve this problem. As a result, I'll be using Pytorch to classified the images.

Problem Statement

The goal I build a ML model that can classify dog images with over 60% accuracy also such model can be deploy as endpoint furthermore use inside any web app process real-world.

The algorithm should detect Dog and Human face detector:

- ***Dog face detector:*** Given an image of a dog, the algorithm will identify an estimate of the canine's breed.
- ***Human face detector:*** If supplied an image of a human, the code will identify the resembling dog breed.

Metrics:

The data is split into train, test and valid dataset. The model is trained using the train dataset. The test data is use to predict the performance of the model on unseen data while training.

I'll use test_accuracy as a metric to evaluate our model on valid data comparing the all prediction with the real valid dataset.

test_accuracy = 100*np.sum(np.array(tensor_predicted)==np.array(targets))/len(tensor_predicted)

Data Exploration:

Input format are images those are inside a folder called dogImages, images. The data contains dog as well as human pictures.

In total are 8351 dog images distribute as training (6,680 Images), test (836 Images) and valid (835 Images) directories. Each of the directories has 133 folders each one correspondent to a different dog's breed.

The human data contain 13233 pictures separated by names, each name correspondent to a different folder in total are 5750 folders. The data is not well balance due to the fact they are folder with 1 image while others containt more than 1, the size is 250X250.

Algorithms and techniques:

To solve this multiclass classification, challenge, I'll use Convolutional Neural Network. A **Convolutional Neural Network (CNN)** is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.

The solution involves three steps.

- First, to detect human images, we can use existing algorithm like OpenCV's implementation of Haar feature based cascade classifiers.
- Second, to detect dog-images we will use pretrained VGG16 model.
- Finally, after the image is identified as dog/human, we can pass this image to an CNN model which will process the image and predict the breed that matches the best out of 133 breeds.

Benchmark

- The CNN model created from scratch must have accuracy of at least 10%. In order to confirm that the Pytorch model chooses (NET) is learning.
- The transfer learning model should be accuracy more than 60 %.

Data preprocessing:

- Train data:
    a. Resize to 258 * 258.
    b. Center Crop 244.
    c. Normalize to 0.5 in all edges.
    d. Ramdon Rotation 30.
    e. Horizontal flip.
- Test data:
    - Resize to 258 * 258.
    - Center Crop 244.
    - Normalize to 0.5 in all edges.
- Valid data:
    - Resize to 258 * 258.
    - Center Crop 244.
    - Normalize to 0.5 in all edges.

Note: I want to work with the Pytorch SDK provided by AWS on how to work with custom Pytorch model. In order to do it we must upload the data to S3 at this point I have two options. One uploaded the pictures, second uploaded the tensor after processing the data. The most obvious answer is the second option but I have problems importing .pt data to the train.py file so after several days if train and failing it, with different configurations I decided to choose the first option.

Implementation:

The model choose has 3 convolutional layers with kernel size of 3 and stride of 2. The first conv layer (conv1) takes the 258*258 input image and the final conv layer (conv3) produces an output size of 128. ReLU activation function is used here. The pooling layer of (2,2) is used which will reduce the input size by 2. We have two fully connected layers that finally produces 133-dimensional output. A dropout of 0.25 is added to avoid over overfitting.

```
class Net(nn.Module):
    def __init__(self, input_features, hidden_dim, output_dim):
        super(Net, self).__init__()
        ## Define layers of a CNN
        self.conv1 = nn.Conv2d(3, 32, 3, stride = 2, padding = 1)
        self.conv2 = nn.Conv2d(32, 64, 3, stride = 2, padding = 1)
        self.conv3 = nn.Conv2d(64, 128, 3, padding = 1)

        # max pooling layer
        self.pool = nn.MaxPool2d(2, 2)

        # Fully-connected layers
        self.fc1 = nn.Linear(7 * 7 * 128, 500)
        self.fc2 = nn.Linear(500, num_classes)

        #Dropout
        self.dropout = nn.Dropout(0.25)

    def forward(self, x):
        ## Define forward behavior
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = F.relu(self.conv3(x))
        x = self.pool(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
```

Fig 1: Net Model.

Implementation:

Net model: The best accuracy trained model that I got was 25 % but I trained the model per 300 epochs. At this point I decided not to continuing due to the cost of training the model on AWS. The best accuracy was with Lr=0.005 and the EC2 chooses was ml.m4.4xlarge. The CNN submitted in this folder have accuracy of 16%, as explain before it meets the benchmarking, but the model can be improved by using transfer learning and increasing the number of epochs.

To create CNN with transfer learning, I have selected the Resnet50 architecture, the architecture is 50 layers deep. The last convolutional output is fed as input to our model. We only need to add a fully connected layer to produce 133-dimensional output (one for each dog category). The model performed extremely well when compared to CNN from scratch. With just 5 epochs, the model got 43% accuracy.

```
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=2048, out_features=133, bias=True)
)
```

Fig 2. Fully connected layer at the end of the transfer learning model.

Model Evaluation and Validation

Human Face detector: The human face detector function was created using OpenCV's implementation of Haar feature based cascade classifiers. 100% of human faces were detected in first 50 images of human face dataset and 10% of human faces detected in first 50 images of dog dataset.

Dog Face detector: The dog detector function was created using pre-trained VGG16 model. 100% of dog faces were detected in first 100 images of dog dataset and 0% of dog faces detected in first 100 images of human dataset.

CNN using transfer learning: The CNN model created using transfer learning with ResNet50 architecture was trained for 5 epochs, and the final model produced an accuracy of 42% on test data. The model correctly predicted breeds for 356 images out of 835 total images.

Accuracy on test data: 42% (356/835).

Justification

I think the model performance is better than expected. The model created using transfer learning have an accuracy of 42% compared to the CNN model created from scratch which had only 13% accuracy. Even with 5 epochs got twice of the accuracy when I trained the Net per over 300 epochs.

Improvement

- Adding more training and test data, currently the model is created using only 133 breeds of dog.
- Appling more augmentation steps.
- Pictures with more than one dog or person.
- Pictures with the lateral dog faces.
- Applying more Epochs.

References

1. *Original repo for Project - GitHub: https://github.com/udacity/deep-learning-v2-pytorch/blob/master/project-dog-classification/*
2. *Resnet101:*
3. *https://pytorch.org/docs/stable/_modules/torchvision/models/resnet.html#resnet101*
4. *Imagenet training in Pytorch:*
5. *https://github.com/pytorch/examples/blob/97304e232807082c2e7b54c597615dc0ad8f*
6. *6173/imagenet/main.py#L197-L198*
7. *Pytorch Documentation: https://pytorch.org/docs/master/*
8. *https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-*
9. *networks-the-eli5-way-3bd2b1164a53*
10. *http://wiki.fast.ai/index.php/Log_Loss*