MACHINE LEARNING ENGINEER NANODEGREE

CAPSTONE PROJECT REPORT

DOG BREED CLASSIFIER USING CNN

CYNTHIA CLINTON

MAY 23, 2021

Definition

Project Overview:

Identifying dog breeds is a complex challenging task that humans find difficult to accomplish without help from machine learning and computer vision. It involves image recognition and classification that use machine learning techniques to assist computer vision.

A machine learning approach to image classification, a supervised learning problem entails defining, identifying, and obtaining key features from images and using them as input to a machine learning model. Early computer vision models relied on raw pixel data as the input to the model. Also, image classification is a machine learning method, and it is designed to resemble the way a human brain works. A breakthrough in building models for image classification came with the discovery that a Convolutional Neural Networks which is an architecture of a neural network that is commonly used for image classification could be used to extract higher and higher-level representations of an image content. Instead of preprocessing the data to derive features like textures and shapes, a CNN takes an image's raw pixels data as input and learns how to extract these features and identify what object they constitute.

For this project, I have built a Convolutional Neural Networks (CNNs) model from scratch, and I have also built a Convolutional Neural Networks (CNNs) model using transfer learning that can be used within a web or mobile app to classify different breeds of dogs and to classify humans as different breeds of dogs from images. When given an image of a dog and a human, the algorithm will identify an estimate of the canine's breed, or the code will identify the resembling breed.

I have accomplished this task by using pretrained weights from Resnet-101 model, and a Convolutional Neural Network built from scratch that were trained to classify input images into their accurate dog breeds. The data used was provided by Udacity.

Problem Statement:

To identify a dog's breed from a given set of images of dogs and humans, I created a model with good accuracy. The goal of the project is to answer two questions: What dog breed is on the pictures of various dogs and what most similar dog breed will an image of a human represent. In addition, an accuracy of 60 percent or more should be achieved. To accomplish this task, I created a machine learning model where transfer learning with more than 60% of accuracy was used to predict an estimate of the breed of the dog based on a picture of a dog or a human.

The algorithm performed two tasks:
It used a dog face detector to identify an estimate of a dog's breed from a dog's image, and it used human face detector to identify the most similar dog breed from a human's image.

Metrics:

The goal is to correctly predict the breed of a dog's image. Therefore, I used accuracy on the test data, which is how many times the model predicts the correct breed of the dog as an evaluation metric to improve the model after each iteration. If a dog is detected in the image, the accurate predicted breed is displayed. If a human is detected in the image, the accurate resembling dog breed to that human's image is displayed. And if neither dog nor human is detected in the image, an accurate provided output that will indicate that an error has occurred is displayed. The formula used to derive the accuracy is:

$$\text{Accuracy\%} = \frac{\text{total number of correct predictions}}{\text{total number of predictions' images}}$$

Analysis

Data Exploration:

In this section, the datasets that is comprised of dogs and human images are explored and analyzed to identify dogs' breeds.

The datasets and inputs images are provided by Udacity. The input format is of the image type. They are divided into the dog dataset and the human dataset:

The dog dataset is comprised of 8,351 dog images which are subdivided into training, test, and validation data. There are 6,680 images to train the model, 836 images for testing and 835 images for validation.

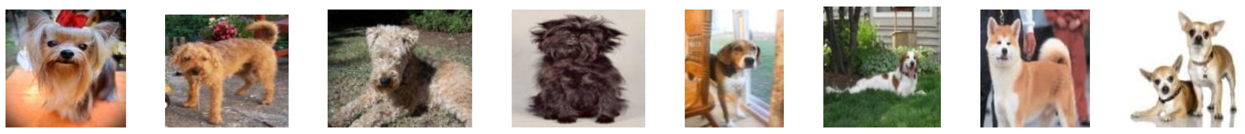
The entire dataset contains 133 different dog breeds. The images sizes and backgrounds are of different sizes and backgrounds. They are not the same. The data will not be balanced. Some breeds of dogs will have 4 images while the other breeds will have 8 images. The lightning is not the same which is totally fine because the model will work on different types of images.

The human dataset is comprised of 13,334 images and 5750 folders. These images are used to test the performance of the human face detector. All the image sizes are 250 x 250. The images will have different backgrounds and different angles, lightning and some images will have more than one face (human or dog) on the image.

Exploration Visualization:

Below are some sample images of dogs and humans from the dog and human datasets provided by Udacity:

Sample Dogs Images Provided by Udacity:



Sample Human Images Provided by Udacity:

Algorithms and Techniques:

To solve the problem of identifying the breed of a dog, I have used Convolutional Neural Networks (CNN), a deep learning algorithm which analyzes visual imagery to design a model that will estimate a possible dog breed from pictures of dogs and humans. This technique involved five steps:

- Used OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images. The OpenCV provides many pretrained face detectors stored as XML files. I used pretrained frontalface XML as the only input in the CascadeClassifier () parameter.

- Used a pre-trained VGG16 model to detect dogs in images of both humans and dogs which has been trained on the ImageNet dataset that is used for image classification. I used the pre-trained VGG-16 model to obtain index corresponding to predicted ImageNet class for the image at the image path. Wrote a function that accepts a path to a 224 x 224 preprocessed normalized image tensor as input and returned the index corresponding to the ImageNet class that is predicted by the pre-trained VGG-16 model. The concept here is that if the output of the model lies within the range of 151-268, it can be assumed that the image has a dog in it. No training was necessary because I used pre-trained weights.

- Passed the images into a Convolutional Neural Network CNN built from scratch and used a CNN with Transfer Learning that processed the images and predicted the breeds that match the images:

- Built the dog breed classifier from scratch using PyTorch to design the model. Then the model was trained, validated, and tested.

- Built the dog breed classifier using Transfer Learning. I used PyTorch to design the Resnet101 pretrained model. The model was also trained, validated, and tested. The training parameters included the number of epochs, the batch size, the learning rate, total number of layers in the network, the total number of nodes in each hidden layer. Used the Stochastic Gradient Descent(SGD) algorithm to train the model.

- Finally, when running the application, it tested the algorithm to determine whether the image contains a human, dog, or neither. If a dog is detected in the image, the most similar dog's breed is returned.   In addition, if a human is detected in the image, the resembling dog breed is returned. If neither a dog nor a human is detected in the image, an error message will be returned.

Benchmark:

The Convolutional Neural Network (CNN) model created from scratch with an accuracy of at least 10% and the pretrained ResNet101 model with at least 60% accuracy were used as the benchmark models for this project. This baseline solution is enough to confirm that the problem is solvable and that the final model will work.

Methodology

Data Preprocessing:

I have applied normalization to all three datasets. Then I used RandomResizedCrop, RandomHorizontalFlip, RandomRotation to augment the images in the train data to prevent overfitting. Data augmentation improves generalization when training the model. These transforms were used to reduce and enhance the performance of the images in the model. The training data images were resized to (224, 224). The images in the valid_data sets were resized to 256 and center croped to 224 X 224. These images were not augmented in the valid_data because it is only used for validation check. I also did not augment the images in the test data set since it is just used for testing. The images in the test data set were also resized to (224, 224) because I noticed that most pretrained models take in that size. This was the only transform tool used in the test data set.

Moreover, the images in the Resnet101 model were also reduced to 224x224 and normalized.

Finally, all the images are converted into tensor before passing into the model.

Implementation:

The implementation of the algorithms and techniques is broken down into five (5) stages:

1. Detect Humans:

I used OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images. I extracted the pre-trained face detector, using the OpenCV's CascadeClassifier method. Loaded a color image using the cv2.inread method. Then converted the color image to grayscale. I found faces in the images using the detectMultiScale method. Added bounding boxes to the images. Finally, after converting the images to grayscale, I used the face_detector function to return a boolean value if human faces were or were not present in the images.

2. Detect Dogs:

I used the pre-trained VGG16 model to detect dogs in images. I loaded the images from the image path. Then I have normalized and transformed the images in the VGG16 model to 224x224. The images were converted to a PyTorch tensor. The model was moved to GPU. I got the prediction for the model. In the prediction's function - VGG16_predict, the transformations were implemented, and the images were fed into the model. There are 1000 categories in the record. The indexes between 151 and 268 are the dog categories. The function dog_detector evaluates the index and returns true or false if a dog is present or not.

3. Create a CNN to Classify Dog Breeds (from Scratch):

In this stage, I have built a Convolutional Neural Network from to help solve the problem. I used PyTorch to design, train and test the model from scratch.

I wanted a simple model that would give me at least 10% test accuracy. Therefore, I constructed the model with three CNN layers. Each of the CNN layers has a kernel size of 3, specifying the height and weight of the convolutional kernel. The stride was set to 1. Padding is set to 0 to add borders. There is a maxpooling with a 2x2 kernel and a stride of 2. This will downsize the image input size by 2. Flatten Layer is used to convert the pooled feature maps to a single vector. There are two fully-connected layers. One is placed before dropout. The dropout of 0.25 is used to avoid overfitting. The final fully-connected layer produces the final output_size of 133 which predicts classes of breeds.

Step 1: Convolutional Layer with 3 input dimensions and 32 ouput dimensions and kernel_size=(3, 3), stride=(2, 2), padding=(1, 1)

Relu Activation function

Pooling layer: MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

Relu Activation Function

Step 2: Convolutional Layer with 32 input dimensions and 64 output dimensions and kernel_size=(3, 3), stride=(2, 2), padding=(1, 1)

Relu Activation Function

Pooling Layer: MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

Step 3: Convolutional Layer with 64 input dimensions and 128 ouput dimensions and kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

Pooling Layer: MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

Flatten Layer to convert the pooled feature maps to a single vector

Fully Connected Linear Layer: Linear(in_features=100352, out_features=512, bias=True)

Dropout with a probabilty of 0.25

Fully Connected Linear Layer: Linear(in_features=512, out_features=133, bias=True)

4. Create a CNN to Classify Dog Breeds (using Transfer Learning):

For this stage, I used transfer learning to build the model. I loaded the pretrained Resnet101 model from PyTorch and froze the feature parameters by setting the requires_grad to false. Then I extracted the fully connected layers, replaced the out_ features of the fully-connected layer with 133 to produce the dog breeds of 133 and set requires_grad to true. The model was trained for 50 epochs.

5. The Final Application:

Within the final app, the Detect Humans, Detect Dogs, Create a CNN to Classify Dog Breeds (from Scratch), Create a CNN to Classify Dog Breeds (using Transfer Learning) are integrated and implemented. The detectors check to see if the image is a dog or human. If a dog and a human are not present in the images, an error message is returned to the users. The image is passed to the predict_breed_transfer function. The images are transformed and normalized. Then a prediction is made using the dog breed classifier model. Finally, the prediction is produced to the user as the predicted dog breed or the most similar dog breeds to the human in the images.

Refinement

I have refined and enhanced the CNN model built from scratch by using transfer learning for better accuracy and performance. I used the Resnet101 architecture which is pre-trained on ImageNet dataset. After using transfer learning, the model performance improved with 80% accuracy meeting the benchmark's expectation.

Results

Model Evaluation and Validation:

I have built a model from scratch as a benchmark model and a final model using transfer learning. Both models were trained, validated, and tested. During the training of both models, the models were validated with the validation set after each epoch and after training, the models were also tested with a test dataset. The best performing hyperparameters were chosen from all the training sets.

The final model using transfer learning was built using the RestNet101 pre-trained model. It performs better than the other models in the top-1 error comparison chart. The architecture of this model where the out_ features of the fully-connected layers were replaced to 133 dog breed classes to produce the dog breeds of 133, was trained for 40 epochs. The final model's accuracy level was 80% on the test data. It correctly predicted 681 dog breeds images out of the test data's 836 images. The learning rate of 0.001 and stochastic gradient descent (SGD) optimizer were used. The model was trained for 40 epochs. This hyperparameter worked well for this image classification job. The test result of 80% accuracy meets the expectation of the final CNN model which must attain at least 60% accuracy on the test set. To verify the robustness of the model, the final model was tested with 836 images comprising of all the 133 dog breeds.

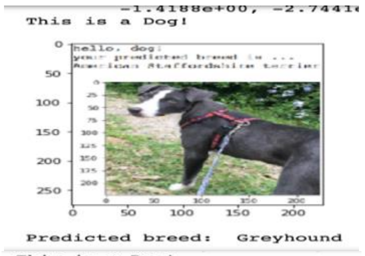
Justification:

The final model has an accuracy of 80% on the test dataset which is a better performance than the expectation of the benchmark of 60%. This is a reasonable enough accuracy that can be used in the final dog breed classifier app.

Conclusion

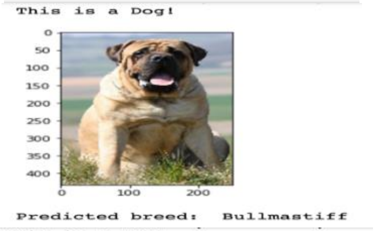
Free-Form Visualization:

These are some of the predicted images that were produced after the model ran:

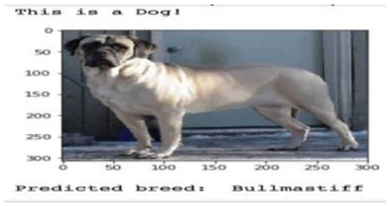


```
-1.4188e+00, -2.7441
This is a Dog!
```

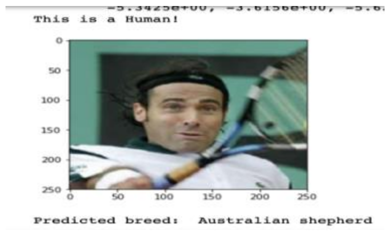
```
Predicted breed:  Greyhound
```



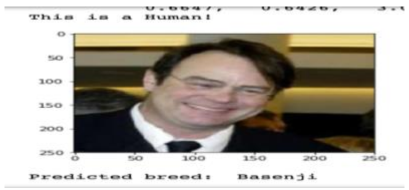
```
This is a Dog!
Predicted breed:  Bullmastiff
```



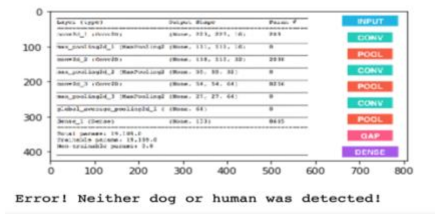
```
This is a Dog!
Predicted breed:  Bullmastiff
```



```
-5.3425e+00, -3.6156e+00, -5.6
This is a Human!
Predicted breed:  Australian shepherd
```



```
0.6647,      0.6426,      3.
This is a Human!
Predicted breed:  Basenji
```



```
Error! Neither dog or human was detected!
```

Reflection:

The following steps are my reflection of what was done in this project:

1. The data: dog and human images were imported into the project.
2. The human face detector was developed and implemented
3. The dog detector was developed and implemented
4. A benchmark was chosen for the model
5. The images were preprocessed, and the data were transformed and augmented.
6. A dog breed classifier was created from scratch, trained, validated, and tested.
7. A final dog breed classifier was created using transfer learning, trained, validated, and tested.
8. All parts of projected were integrated into one cohesive project.
9. All functions have been documented.
10. Created a GitHub repository for the project.

One interesting aspect of the project was building the model from scratch. Setting up the convolutional neural networks architect was exciting and intriguing. Therefore, I was able to create three CNN layers to help solve the problem of classifying a dog breed.

The most difficult aspect of the project was building the model using transfer learning. I was careful in choosing the right pretrained Resnet model to use to produce the best results.

Improvement:

The model can be improved by using different transfer learning architectures for extracting fully connected layers. In addition, the number of dog breeds classified can be increased to a suitable amount. More training and test data sets could be added and more image transformations and augmentations for better performance of the model could be implemented. Fine tuning the optimizer’s parameters could also generate better results. The learning rate could also be adjusted for good training and test results.

References:

https://en.wikipedia.org/wiki/Convolutional_neural_network/

https://machinelearningmastery.com/how-to-know-if-your-machine-learning-model-has-good-performance/

https://opencv.org/

https://github.com/udacity/deep-learning-v2-pytorch/blob/master/project-dog-classification/

https://iq.opengenus.org/basics-of-machine-learning-image-classification-techniques/

https://pytorch.org/docs/master/