Machine Learning Engineer Nanodegree

Capstone Project | **CNN Project: Dog Breed Classifier**

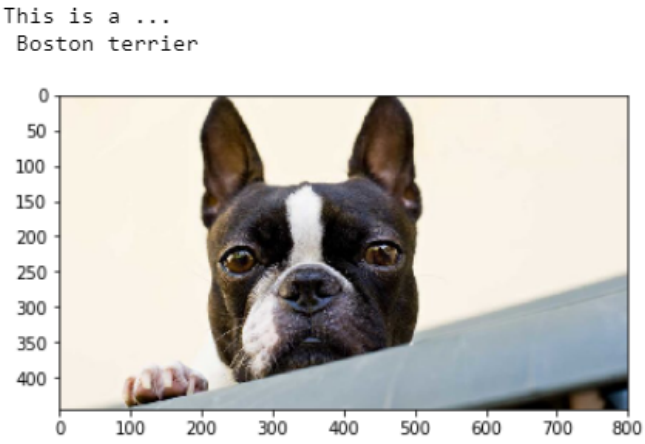
Sedat Erkal

July 31st, 2020

I. Definition

Project Overview

In recent years, software developers will need to know how to incorporate deep learning models into everyday applications. Anything with a camera will be using image classification, object detection, and face recognition, and so on, all based on deep learning models. In this project, it is aimed that the proposed model will accept any user-supplied image as input. If a dog is detected in the image, it will provide an estimate of the dog's breed. If a human is detected, it will provide an estimate of the dog breed that is most resembling. The image below displays the potential sample output of the finished project.



The related dataset consists of human and dog images in order to use in the classification task. Human images are used to detect human faces and give resembling dog breed in the prediction phase. Similarly, dog images are used to detect dogs, train the model, and give the predicted breed. Dog images dataset is particularly helpful to detect dogs in images since we use a pre-trained model (VGG-16) that has been

trained on ImageNet[1], a very large, very popular dataset used for image classification and other vision tasks.

Problem Statement

The task of assigning breed to dogs from images is considered exceptionally challenging. To see why consider that even a human would have trouble distinguishing between different breeds. In this real-world setting, we will need to piece together a series of models to perform different tasks; for instance, the algorithm that detects humans in an image will be different from the CNN that infers dog breed. Given an image of a dog, the proposed algorithm will identify an estimate of the dog's breed. If supplied with an image of a human, the code will identify the resembling dog breed. One alternative solution to the problem is benefiting from the pre-trained models that lead to obtaining higher accuracies.

Metrics

Due to dealing with a classification task, the accuracy score is a good evaluation metric for our specific case both in the benchmark model and the solution model. Accuracy is one of the ways to measure how good a model is. It is the ratio between the number of correctly classified points and the number of total points. The formula for accuracy is

$$\text{Accuracy} = \frac{\text{Correctly classified points}}{\text{All points}}$$

For the benchmark model, it is supposed to have at least an accuracy of 10% and for the solution model, it is ensured that the accuracy is higher than 60%. Accuracy may not always be the best metric to use. It is pretty tricky to just look at accuracy and use that to evaluate our model because it may completely miss the point when the data is skewed. It is a suitable metric for this problem; however, we will look up the F1 score as well, since our data is slightly imbalanced, therefore F1 score would show us more realistic and interpretable results. The formula for F1-Score is

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

[1] http://www.image-net.org/

II. Analysis

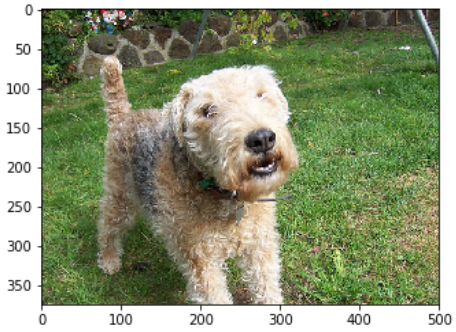
Data Exploration

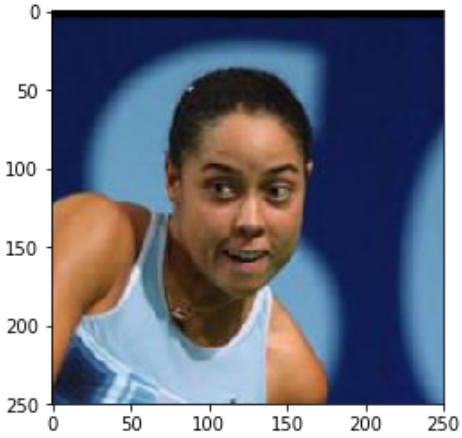
For this project, the input format is image type, because we want to input an image and identify the breed of the dog. The dataset has pictures of dogs and humans. The datasets we will use are provided by Udacity with 133 dog breeds. There are 13233 total human images and there are 8351 total dog images for dog classification. Also, there is a class imbalance in the training dataset since some classes have fewer images while others have more.

The dog dataset is used for training, validation, and testing of the classifier. The human dataset is used for testing the dog breed classifier in such a case that it predicts which breed a human looks like.

Exploratory Visualization

The human images have the same dimension 250×250. The dog images have various dimensions and will be resized to 224×224. Here are some sample images from the datasets.





Algorithms and Techniques

The solution requires the multiclass image classification. Since the classified objects are images, convolutional neural networks (CNNs) are appropriate candidates for the current task. Starting with detecting humans and dogs in images, we need a way to predict breed from images. Our vision-based algorithm will have to conquer high intra-class variation to determine how to classify different shades as the same breed. That's why we will use transfer learning to create a CNN architecture in the classification task that attains greatly improved accuracy.

CNNs are the type of Deep Learning algorithms that can take tensors (e.g. batches of images) as an input, establish the relations between image features by learning weights and biases, output a predicted class and learn from an error. Algorithms and techniques used in the project are

- Human Face Detector - by OpenCV's implementation of pre-trained Haar feature-based cascade classifiers
- Dog Detector - by trained on ImageNet VGG-16 model
- CNN to Classify Dog Breeds (from Scratch)
- CNN to Classify Dog Breeds (using Transfer Learning)

Benchmark

The task of assigning breed to dogs from images is considered exceptionally challenging. To see why consider that even a human would have trouble distinguishing between different breeds. Our proposed approach to this task will be using a CNN from scratch that classifies dog breeds and attains a test accuracy of at least 10%. This can confirm that the model is working because a random guess will provide a correct answer roughly 1 in 133 times, which corresponds to an accuracy of less than 1%.

```
## Define layers of a CNN

# CNN Layers
self.layer1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
self.layer2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
self.layer3 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
self.layer4 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
self.layer5 = nn.Conv2d(128, 256, kernel_size=3, padding=1)

self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

# FC Layers
self.fc1 = nn.Linear(7*7*256, 2048)
self.fc2 = nn.Linear(2048, 1024)
self.fc3 = nn.Linear(1024, classes)
self.dropout = nn.Dropout(0.2)
```

With the given model architecture (above), the benchmark model results in a test accuracy of 16% that is a better performance than expected. However, with the following application (transfer learning), this result will be outperformed.

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

```
Test Loss: 3.552363

Test Accuracy: 16% (138/836)
```

III. Methodology

Data Preprocessing

The input images were resized to (3, 224, 224) since it is the size of images from ImageNet and is expected by VGG models. The training, test, validation images are resized and center cropped, then randomly flipped in the horizontal direction before transforming into tensors. The transformed data are organized into train, test, and validation directories, respectively.

```
# Define transforms for the training, validation, and testing sets
train_transforms = transforms.Compose([transforms.RandomRotation(30),
                                       transforms.RandomResizedCrop(224),
                                       transforms.RandomHorizontalFlip(),
                                       transforms.ToTensor(),
                                       transforms.Normalize([0.485, 0.456, 0.406],
                                                            [0.229, 0.224, 0.225])])

test_transforms = transforms.Compose([transforms.Resize(255),
                                      transforms.CenterCrop(224),
                                      transforms.ToTensor(),
                                      transforms.Normalize([0.485, 0.456, 0.406],
                                                           [0.229, 0.224, 0.225])])

# Load the datasets with ImageFolder
train_data = datasets.ImageFolder(train_dir, transform=train_transforms)
valid_data = datasets.ImageFolder(valid_dir, transform=test_transforms)
test_data = datasets.ImageFolder(test_dir, transform=test_transforms)

# Using the image datasets and the transforms, define the dataloaders
trainloader = torch.utils.data.DataLoader(train_data, batch_size=64, shuffle=True)
validloader = torch.utils.data.DataLoader(valid_data, batch_size=64, shuffle=True)
testloader = torch.utils.data.DataLoader(test_data, batch_size=64, shuffle=True)
```

Implementation

Instead of training a deep neural network from scratch, which would require a significant amount of data, power, and time, it's often convenient to use a pre-trained model and just finetune its performance to simplify and speed up the process. Here, we transfer a CNN model: ResNet-18[2] to improve feature engineering and transfer learning for the dog breed classification. Implemented additional fully-connected dense layers in order to adjust the architecture to the current problem.

```
# Specify model architecture
model_transfer = models.resnet18(pretrained=True)

# Freeze the parameters so we don't backprop through them
for param in model_transfer.parameters():
    param.requires_grad = False

model_transfer.fc = nn.Sequential(nn.Linear(model_transfer.fc.in_features, 1024),
                                  nn.ReLU(),
                                  nn.Dropout(0.2),
                                  nn.Linear(1024, classes))
```

Refinement

Several structures have been tested with both benchmark and solution CNN models. With the benchmark model different low accuracy scores obtained by adjusting different hyperparameters such as number of epochs, kernel size, learning rate, number of hidden units, batch size, etc. However. I conclude that the structure of scratch CNN is too basic for achieving higher accuracy. Therefore, using a more established pre-trained model becomes a necessity in order to achieve better results. So, ResNet-18 pre-trained model is used to serve as the real dog breed classifier without needing many adjustments, an accuracy score of 82% is achieved.

IV. Results

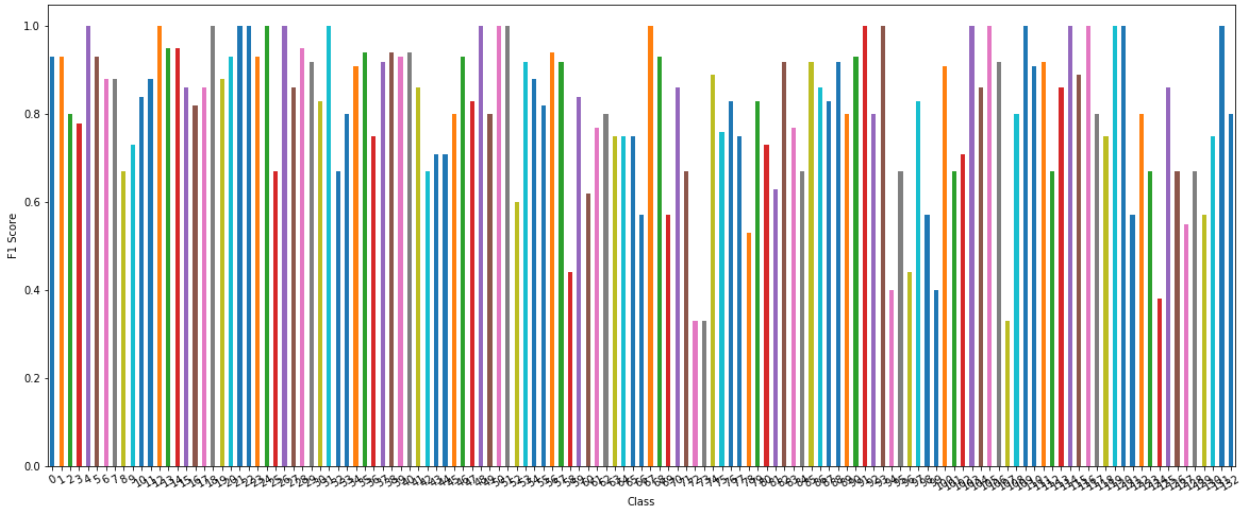
Model Evaluation and Validation

During development, a validation set was used to evaluate the model. The validation set is used for hyperparameter tuning and the test set is used to test the performance of the model. After completing the training and testing of the model we achieve an accuracy score of 82% (692/836).

[2] https://www.kaggle.com/pytorch/resnet18

The parameters used to conduct the model are common choices for such a problem such as using a dropout value of 0.2 for regularization, Adam as an optimizer with a learning rate of 0.001, etc. In addition, freezing the parameters of the pre-trained model so we don't backprop through them and only train the classifier parameters.

Furthermore, from the F1 score plot (below) of all tested data, we found that a lot more breeds of dogs have the highest F1 score, indicating the transfer learning model is more robust for different dog breeds.



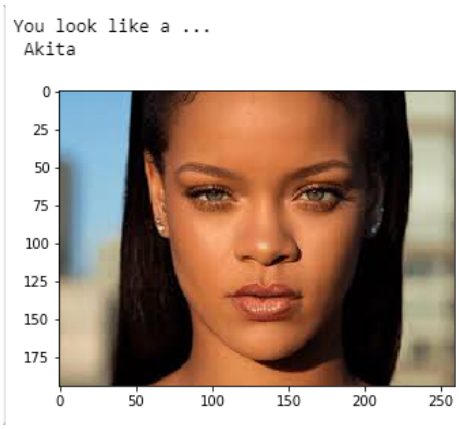
Justification

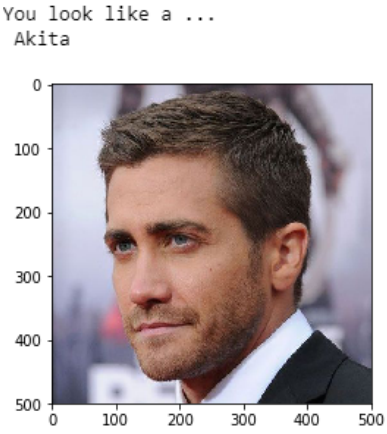
It is expected that the solution model outperforms the benchmark model since the benchmark model is conducted with intuitions and naive assumptions. For our task, it is clear that the solution model provides higher performance than the benchmark model. With the help of transfer learning which uses a pre-trained model that provides weights especially for feature detection, increases the performance of the classifier. It can be said that the final solution is capable of the dog breed classification problem.

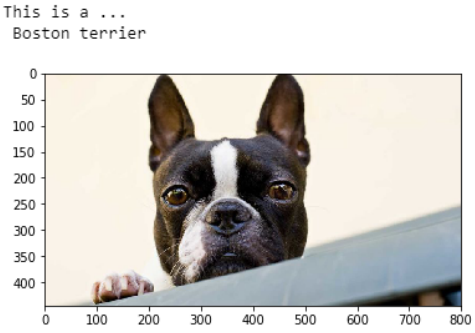
V. Conclusion

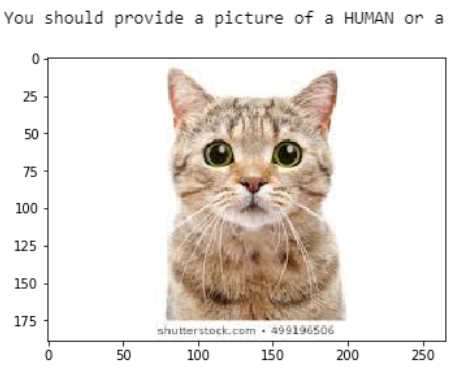
Free-Form Visualization

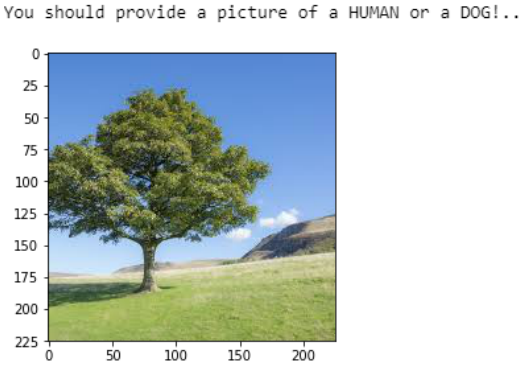
The algorithm is tested with sample images that contain a dog, human, cat, and tree.











Reflection

The end-to-end implementation of the dog breed classifier is conducted in 7 steps:

1. We start by importing the relevant human and dog images into folders.
2. We use OpenCV's pre-trained face detector in order to detect human faces.
3. We use the pre-trained VGG-16 Model with the aim of detecting dogs.
4. We create a CNN architecture from scratch as a benchmark model to classify dog breed.

5. We create another CNN architecture but this time with the help of transfer learning as a solution model to classify do breed that achieves better performance.
6. We create an algorithm that accepts a file path to an image and first determines whether the image contains a human, dog, or neither. Then, if a dog is detected in the image, the algorithm returns the predicted breed, if a human is detected in the image, the algorithm returns the resembling dog breed, and lastly, if neither is detected in the image, the algorithm provides an output that indicates an error.
7. We conclude with testing the algorithm on sample images.

The hardest part of the project is step 4 since that is the state-of-the-art to find the suitable optimizer and the architecture leading to decrease loss on validation at lest to achieve a threshold value.

Improvement

There are possible improvements may be:

- Turning the code into a web app using Flask.
- Using different pre-trained OpenCV face detectors in order to consider different occurrences such as Snapchat-like filter with dog ears on detected human heads.
- Returning not only the highest predicted breed but rather probabilities of top-5 predicted breeds.

These are just a few suggestions and it is clear that the current solution model can become a benchmark model for another solution model since there is always a place for improvement with better architecture, face detector, pre-processing, etc.