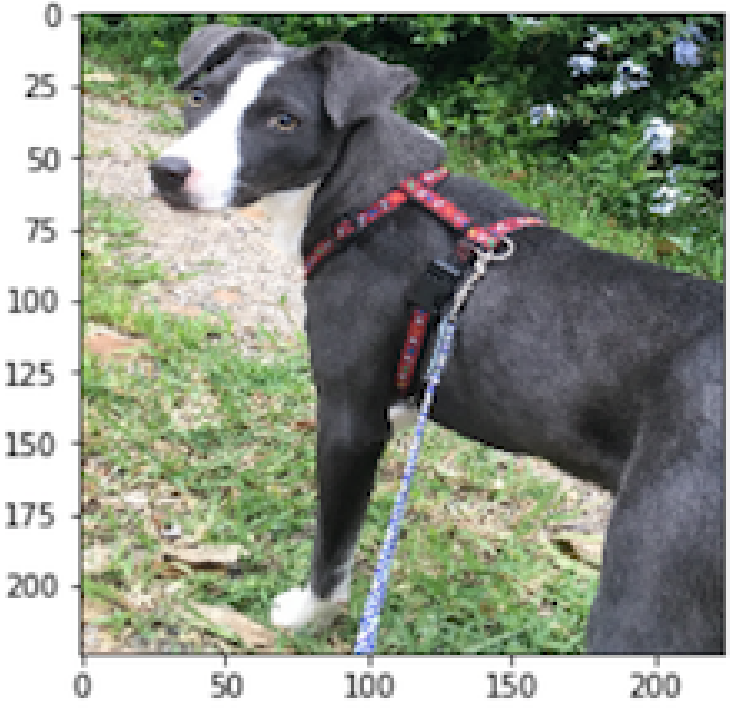
Dog breed classification

Suzanne ten Hage
March 2022

```
hello, dog!
your predicted breed is ...
American Staffordshire terrier
```



Introduction

Humans have been keeping dogs as pets for a very long time. Dogs provide a lot of happiness and companionship. Furthermore, pets are a lovely target for taking pictures, and many pictures of pets, and more specifically dogs, exist on the internet and are being shared amongst family and friends.

There are many different dog breeds and it has become difficult to recognize them all instantly based on their looks. The American Kennel Club recognizes 197 dog breeds globally, of which the most recent addition is the Biewer Terrier in 2021 [1]. Even more, many dogs are a mix of different dog breeds, making it even harder to guess what breed they belong to.

The aim of this project is to create an application that takes dog images as input and outputs the predicted dog breed. In the application, it is also possible to input a human face, after which the application returns the dog breed that has the closest resemblance with the human face.

Our dataset contains 133 dog breeds. In order to be able to tell the breed of any dog, a new application will be developed. The users of the application will be able to take pictures of their dogs, their friend's dogs, or dogs they meet on the street and use this application to classify which breed of dog it is. To make the application even more fun to use, users would be able to know which breed they resemble most closely.

In this report, we start with some data exploration to get to know our datasets. Once we know the data that we are dealing with, we will discuss the methods used to classify human and dog images to dog breeds. Next, we will look at the results/performance of our application. We will finish the report with some conclusions and ideas for future directions.

Data Exploration

Both dog and human images are provided by Udacity and can be downloaded in the dog_app notebook. The zip files are not provided in this submission, as asked in the project description. There are 13233 images of humans and 8351 images of dogs available for this project. The dog images are split into a train, test, and validation set of 6680, 836, and 835 images, respectively. The human images are not split into separate sets, but they also don't have to be since the algorithm won't be trained on the human images at all.

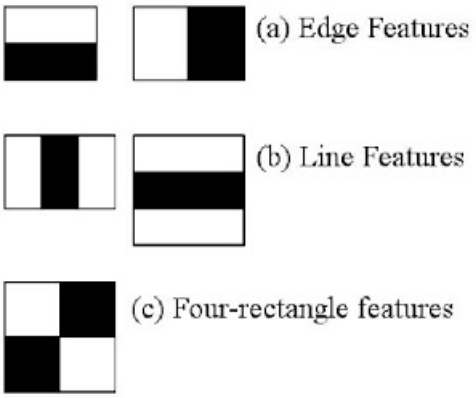
For the use of images in our CNNs, the images need to be transformed. The images are resized to images of 224 x 224 pixels, transformed to tensors and then normalized using following values: mean = [0.485, 0.456, 01.406] and standard deviation = [0.229, 0.224, 0.225]. These values are obtained from the pytorch vgg documentation [2]. Furthermore, for training purpose, the images are randomly rotated by 15 degrees, and randomly flipped horizontally. This creates more diversity in the images.

Methods

Now that we have a more clear understanding of the data that we are working with, we can start building the project. The project contains the five steps; (1) human face detection, (2) dog detection, (3) building a benchmark model, (4) building a more accurate model using transfer learning, and (5) building the app infrastructure. Each step will be outlined in more detail in this chapter.

1. Human face detection

First, we use an OpenCV pre-trained face detector to determine whether or not a picture contains a human face. More specifically, we use the Haar feature-based cascade classifiers model, which is conveniently already downloaded into a folder by Udacity [3]. In this model, a window slides over the image, and in each position it will extract features, using haar filters. These haar filters are shown in the figure on the right.



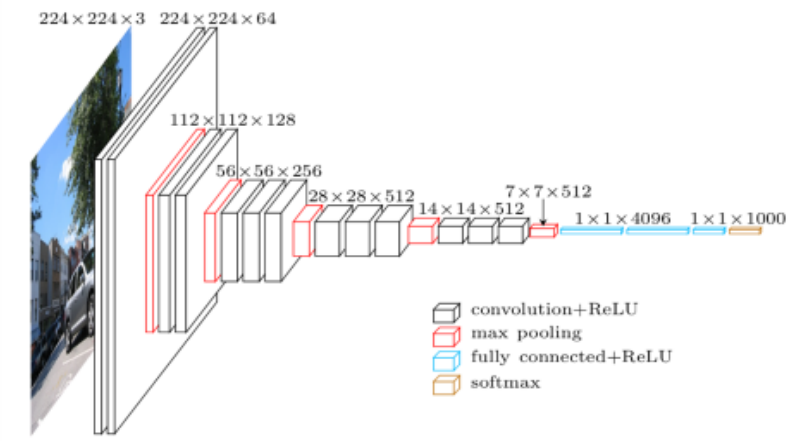
This will create many, many features, after which Adaboost is used to select the most important features. This selection procedure for the most important features is done using a cascade of classifiers. This means, that all the features are grouped into different stages of classifiers, if a window fails the first stage, it is not considered any further. The final classifier to determine if this window contains a face or not is the weighted sum of all the classifiers. Lastly, it is important to note that all images are converted into grayscale before feeding them to the model.

2. Dog detection

Dog detection will be done using the pre-trained model VGG16, available in PyTorch. The pre-trained model is used directly (i.e. without additional training), to determine if a dog is present in the image. The output of the model is an integer indicating an ImageNet category.

The architecture of VGG16 is detailed below and in the figure below. The convolutional layers always have a 3x3 kernel, “same” padding, and ReLu non-linearity. The pooling layers always have a 2x2 pool size and a 2x2 stride.

- 2 convolutional layers of 64 channels
- 1 maxpool layer
- 2 convolutional layers of 128 channels
- 1 maxpool layer
- 3 convolutional layers of 256 channels
- 1 maxpool layer
- 3 convolutional layers of 512 channels
- 1 maxpool layer
- 3 convolutional layers of 512 channels
- 1 maxpool layer
- 3 fully connected (ReLu) layers
- a softmax output layer



3. The benchmark model

Next, a CNN is created from scratch to serve as a baseline model to improve upon. In this model, the convolutional layers always have a 3x3 kernel, 1x1 padding, and ReLu non-linearity. The pooling layers always have a 2x2 pool size and a 1x1 stride. The CNN from scratch architecture is as follows:

- 1 convolutional layer with 16 channels
- 1 convolutional layer with 32 channels
- 1 convolutional layer with 64 channels
- 2 fully connected (ReLu) layers

After the last convolutional layer, and the first fully connected a dropout step is included. In this step, the inputs of random units are set to 0 during training. This is done to prevent overfitting.

The model is trained using the stochastic gradient decent algorithm with momentum implemented and cross entropy loss. In momentum gradient decent the gradient is used to determine accelaration instead of speed, as is the case in regular gradient decent. The advantage is that this algorithm moves faster in valleys, taking less time to converge. The faster momentum can also help prevent getting stuck in local optima. The learning rate is set to 0.01 and the momentum parameter to 0.9.

4. Transfer learning

To get a better accuracy than our CNN form scratch, we adapt the VGG16 model to output a clasification for one of the 133 dog breeds and train this new version using transfer learning. In transfer learning, the weights of a pre-trained model are used as the starting point for training a new model. The model is trained using nesterov momentum optimization, which is very similar to momentum optimization, except that it looks at the momentum slightly ahead, instead of at the local positin, making it slightly more accurate than momentum optimization. The paramaters and loss are kept the same as in the CNN from scratch. All weights, except for the final layer are frozen during training of the new network.

5. An application to recognize dog breeds

Finally, we build a function that takes an image path as input and outputs one of three things; (1) if a dog image is detected, it will output the dog breed of that dog, (2) if a human face is detected it will output the dog breed that human resembles most closely, (3) if no dog or human face is detected, the function will output a message saying “no dogs or humans detected”. The function will also print the image that was put in. The application uses the adapted VGG16 model to make its predictions.

Results and conclusions

Human faces are detected using the Haar feature-based cascade classifiers by OpenCV and evaluated using false positives and false negatives. The negative images in the dataset are all images from the dog dataset. In total, about 2% of the images are false negatives, which means that the image actually contained a human face, but this was not detected by the model. There are 17% false postives, which means that there was no human face in the image, but the model still recognized it as a human face.

The dogs are detected using the original pre-trained VGG16 and evaluated using false positives and false negatives. The negative images in the dataset are all images from the human faces dataset. No false positives or false negatives were detected, meaning that the model perfectly identified all images that contain a dog and also didn't include any images that did not contain a dog.

The CNN from scratch has a performance of ~10% accuracy (i.e. 86 out of 832 test images are classified correctly), while the adapted VGG16 model has a much higher accuracy at 71% (i.e. 599 out of 832 test images are classified correctly). The test accuracy of the dog breed classification is calculated as follows; 100 * correct / total. The final dog breed detection application uses the second model, due to the higher accuracy.

The main reason for the higher accuracy in the VGG16 adapted model is the more sophisticated model architecture and the use transfer learning. A more complex algorithm also requries more computation power to train the model. Transfer learning is a great method to decrease the computation power needed to achieve a goal. In transfer learning, the starting point is much closer to your actual goal, because you start with a model that is able to perform a similar task, instead of starting somewhere randomly. It is therefore often not needed to retrain all layers of the model, leaving especially deeper layers of the model untouched. This saves time and power needed to train the model for the new tasks.

Lastly, the application is also able to detect human faces and give the dog breed that it resembles most closely or output a message if no dogs or humans are detected.

Future directions

The model performance currently sits at 73% accuracy. To improve the model one could play aorund with adapting the vgg16 architecture more, using a different pre-trained model for transfer learning (e.g. resnet50), or freezing less layers during training.

Furthermore, instead of having a notebook function to use the model, it would be interesting to implement the model into an ios/android application. Users should be able to take pictures or upload pictures to the application. The application then calls the model, and outputs which breed the dog or human in the image resembles.

Last, but not least, not everybody is a dog person. To widen the target audience it would be interesting to see if we can expand our model to also automatically recognize cats and classify into cat breeds. In this case, the human faces can be either automatically classified as cat or dog, depending on which is resembled more. Or the application can have a feature where the user can choose between cats/dogs.

References

[1] https://www.akc.org/press-center/articles-resources/facts-and-stats/breeds-year-recognized/ (01/31/2022)
[2] https://pytorch.org/hub/pytorch_vision_vgg/ (24/03/2022)
[3] https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html (11/03/2022)