Dog Breed Identification

Project Report

Grzegorz Lippe

February 25, 2021

Definition

Project Overview

In this project the knowledge and usage of Convolutional Neural Networks (CNN) is demonstrated for the Udacity reviewer team. Within this project a CNN with the ability to detect dog breeds with an accuracy of more than 10 % is created from scratch. Afterwards a pre-trained VGG16[1] [2] is adapted to perform the same task with an accuracy of >60 %.

Besides a neural net for dog breed detection, also an existing OpenCV[3] neural net for face recognition is adapted. The algorithm provides a bounding box for each face it found within a picture.

Both algorithms, the OpenCV face recognition and the VGG16 dog breed detection are combined into an demo app, that can take an arbitrary picture and return a statement of a most likely dog breed. If a picture of a human is provided, the app should identify the human, but still provide a statement of a most similar looking dog breed.

Problem Statement

Given that the human face recognition is a takeover OpenCV network, the problem solved is the identification of a dog breed with the help of a convolutional neural network.

[1] Pytorch Models
[2] Very Deep Convolutional Networks for Large-Scale Image Recognition
[3] Open Computer Vision

Metrics

For the evaluation the labelled data is used. For example the accuracy of the face detection algorithm can be measured by feeding it with (previously labelled) images of humans:

$$Score = \frac{\text{Humans detected}}{\text{Images presented}}$$

A slightly modified algorithm is used to evaluate the dog detecting model. The dog detector provides a breed identification on top, so in the first step the upper term can be used to judge the accuracy of the dog detection. Additionally the quality of the breed identification is measured as follows:

$$Score = \sum_{breeds} \frac{\text{correct breed identified}}{\text{breed provided}}$$

Analysis

Data Exploration

The data consists of two different sets, provided by Udacity and are described as follows:

dog dataset The dog data set is already divided into three different datasets (folders) for test, training and validation. Each folder contains sub directories of the specific formating `DDD.breed_name`, where `DDD` represents a 3 digit number, followed by the breed name after a dot. Each subfolder contains a hand full of images of the specific breed.

Overall there are 133 different dog breeds, and 835 images provided for validation, 6680 images for training and 836 images for testing. The images within the training dataset are unbalanced, the amount of images per breed varies from 26 for the Norwegian Buhund and the Xoloitzcuintli to 77 for the Alaskan Malamute. So some breeds are represented roughly 3 times more often.

The dog images contain a singular dog each, mostly of the whole dog, sometimes only of the snout, but they are not equally sized, they vary from 5 kB up to 5 MB. Their aspect ratios cover all ranges between portrait, landscape and quadratic.

human dataset The human dataset consists of 13233 images of 5749 persons, which are stored in a separate directory named after each celebrity. The images of the people are already cropped and centred around their face and all of the same size of 250x250, but the backgrounds vary. Sometimes there are additional people in the background.

Exploratory Visualization

Human Dataset

Here is a random sample of the human images dataset:

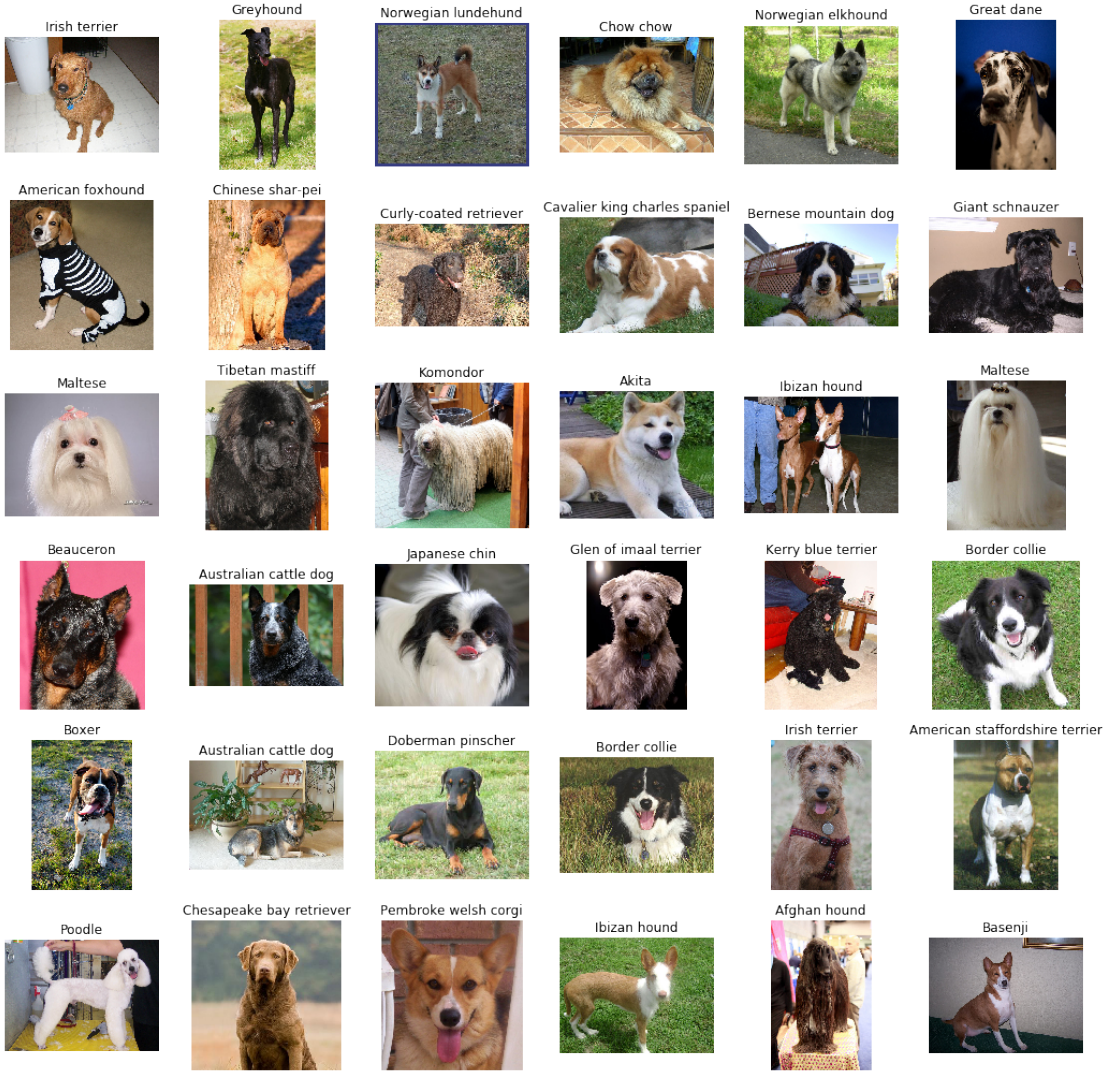
Its worth noticing, that George Bush appeared two times in this randoms sample of 36 images out of 13233, so this is a hint for an unbalanced dataset. A further investigation of the dataset confirms the suspicion, the following bar plot, figure 5 in the appendix, shows the amount of pictures of the top 25.

George Bush, Collin Powell and Tony Blair alone are represented in 910 Images or roughly seven percent of the dataset. On the other hand 4069 celebrities of the 5749 persons are represented by only one image.

If the problem at hand would be the training of a face recognition algorithm this unbalanced dataset would be a problem, but within this project it is only used for sample data sets to test the demonstrator app with later.

Dog Dataset

Here is a random sample of the dog images dataset:



At a first glance everything is fine, there are images of dogs, sometimes whole dogs and sometimes their snouts only. Therefore a more thorough investigation is performed.

The following table shows the amount of breeds (count) for each subset: "train", "test" and "valid". The mean value shows how many images are present in average in every breed: 50.

	train	test	valid	sum
count	133.000000	133.000000	133.000000	133.000000
mean	50.225564	6.285714	6.278195	62.789474
std	11.863885	1.712571	1.350384	14.852330
min	26.000000	3.000000	4.000000	33.000000
25%	42.000000	5.000000	6.000000	53.000000
50%	50.000000	6.000000	6.000000	62.000000
75%	61.000000	8.000000	7.000000	76.000000
max	77.000000	10.000000	9.000000	96.000000

Also it states, that there are at least 26 images for each breed in the training set and 77 at best. A more visual representation about the balance of the data set is provided in the appendix within this document in the bar plot 6.

Additionally also the sizes and aspect ratios of the images matter. These were investigated by extracting these information out of each file and storing them in a Pandas[4] DataFrame for further investigation.

The following figure 1 shows a logarithmic histogram of the file sizes of the dog images. Most images are roundabout 100 kB in size, starting at 4 kB but leading up to 10 MB.

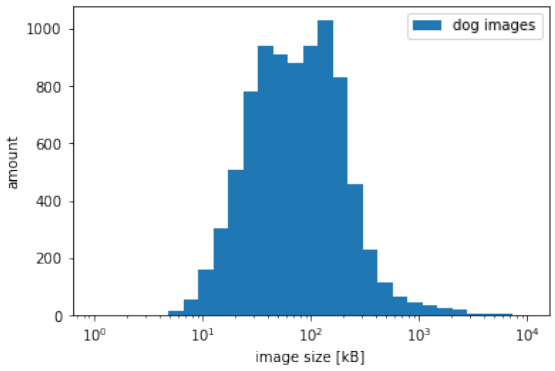


Figure 1: Histogram of the dog image sizes [kB, logarithmic]

The top 5 image sizes were inspected visually and the biggest image in the dataset is some image of two people at a dog show. This image, see figure 2 is not suitable for training and therefore discarded from the dataset.

In the same manner the top and bottom five aspect ratios where investigated. The image with the "highest" Aspect ratio is a montage of two Lhasa Apso dogs, which is fine to use in the test data set.

[4]Panel Data

Figure 2: The maximum sized image and the maximum aspect ration image of the dog dataset

Algorithms and Techniques

The problem of the dog breed identification will be solved using a convolutional neural network. These networks are especially good on solving the tasks of image recognition, within a 2-dimensional space.

Whereas classical neural networks depend on a flat input vector and usually fully connect the different layers, in a convolutional network many different subsets of images are analysed with the help of a so called kernel. The kernel can be thought of as a kind of filter, which extracts the information that pixels have in relation to one another, like for example a horizontal line. This extracted information than is passed on to another layer and can be abstracted even more.

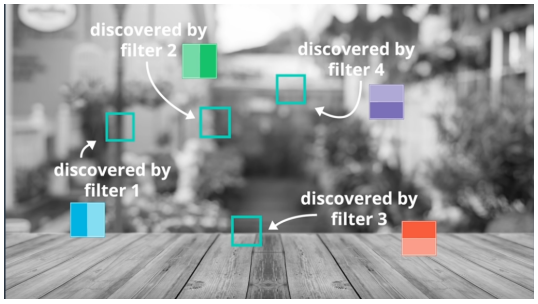


Figure 3: A scheme of how different kernels detect different features, source: Udacity

Figure 3 shows an Udacity example of how four different kernels detect four different kinds of features in an image:

- horizontal lines from dark to light
- horizontal lines from light to dark
- vertical lines from dark to light and

- vertical lines from light to dark.

The whole training of a CNN is the optimization of the weights in the different kernels of the neural net. So the CNN learns to extract different features of images within the training task.

Benchmark

As a benchmark I will use a naive predictor. The dataset is slightly unbalanced so assuming that there are 133 different dog breeds, some of the occurring up to three times more than others, the naive approach should guess the correct breed (b) with $p_b = \frac{k_b}{3 \cdot 133}$, $k_b = [1 \dots 3]$, or less than 1%. The implemented model shall get at least 10%.

Methodology

Data Preprocessing

The images are preprocessed and augmented in three different kinds, for the testing, the training and the validation data set.

Listing 1: Image Transformation

```
# documentation values:
# https://pytorch.org/docs/stable/torchvision/transforms.html
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])

training_transform = transforms.Compose([transforms.RandomResizedCrop
    (256),
                                         transforms.RandomHorizontalFlip(),
                                         transforms.ToTensor(),
                                         normalize])

validation_transform = transforms.Compose([transforms.Resize(300),
                                         transforms.CenterCrop(256),
                                         transforms.ToTensor(),
                                         normalize])

test_transform = transforms.Compose([transforms.Resize(size=(256,256)),
                                         transforms.ToTensor(),
                                         normalize])
```

Listing 1 shows all image preprocessing steps performed for the training of the CNN. All in common for the testing, validation and training data set is a normalization and a transformation into a `torch.transforms.ToTensor()` data type.

The normalization takes place with documented values and changes the representation of the integer RGB values $\underline{n} \in \mathbb{N}^3, [0 \dots 255]$ to normalized float values $\underline{q} \in \mathbb{Q}^3, [0.0 \dots 1.0]$. Two dimensions for the image, the third for the colour.

The test dataset gets an additional resize to a quadratic shape 256^2. If the image was portrait or landscape before it now is squeezed, but that doesn't matter for the recognition task. I chose 256, because it is a power of 2.

The validation dataset undergoes a different transformation. It is first resized and then cropped to the center, because the validation images all contain the dog in the center of the image.

The training dataset a different kind of resize, the `RandomResizedCrop`. This causes the image to be cropped at a random position of the in put image. Therefore the training data may or may not contain an whole dog or just a part of a dog. This leads to a harder training with better results and prevents over fitting. For the same reason the training dataset also is preprocessed with a random horizontal flip.

Implementation

Transfer Learning with a pre trained Network

The adaption of the pre trained VGG16 neural network is thoroughly described in the Udacity Lesson "Transfer Learning" within the extra curricular material[5] for the "Machine Learning Engineer Nano Degree". Essentially the VGG16 network was trained on the image-net data base with 14 million images and can extract 1000 classes within the image-net database. These 1000 labels are defined by the last fully connected layer of the CNN, see figure 4.

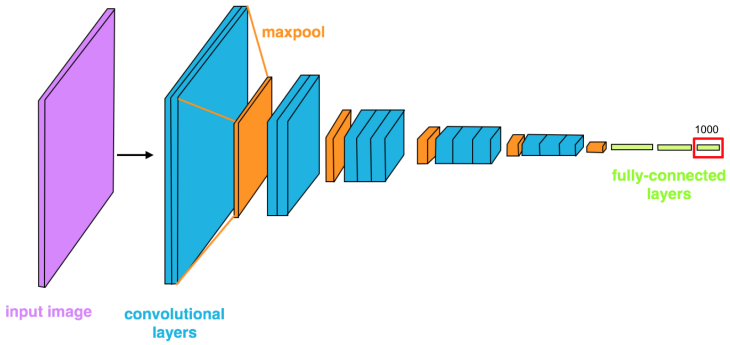


Figure 4: VGG-16 Architecture, Source: Udacity

[5] Transfer Learning Lesson

The benefit of such a highly trained network is, that all the image recognition filters, which are there to recognise a vast amount of different geometrical artifacts like: horizontal lines, vertical lines, circles etc. are already there. Moreover the subsequent pooling and convolutional layers are also trained to identify more complex shapes like: wheels, aeroplanes, or fur.

Therefore the neural net can be adapted and retrained, by only changing the last layer, which defines the 1000 labels of the VGG16. The following listing 2 shows the classifier of the VGG, which correspondent to the three green boxes in figure 4:

Listing 2: Original VGG16 Classifier

```
Sequential(
  (0): Linear(in_features=25088, out_features=4096, bias=True)
  (1): ReLU(inplace=True)
  (2): Dropout(p=0.5, inplace=False)
  (3): Linear(in_features=4096, out_features=4096, bias=True)
  (4): ReLU(inplace=True)
  (5): Dropout(p=0.5, inplace=False)
  (6): Linear(in_features=4096, out_features=1000, bias=True)
)
```

There are three fully connected linear layers, the last one providing 1000 nodes for 1000 image classes. The dog breed data set contains 133 breeds, so the following three lines of code reconfigure the model 3.

Listing 3: Adapting VGG16 Classifier

```
  # Freeze training for all "features" layers
  for param in model_transfer.features.parameters():
      param.requires_grad = False

  model_transfer.classifier[6] =  nn.Linear(4096, 133) # from the flower
      exercise I knew there are 4096 input nodes in the second to last
      layer
```

The fist two lines of code tell the model to not adapt (require gradients) within the training process. With that option the already existing and capable neural net will be preserved during training. The last line changes tha last fully connected layer to provide 133 exit nodes. These are trainable and will be adapted within the training process.

After that the model is provided with a loss function (`torch.nn.CrossEntropyLoss`) and a optimization function (`torch.optim.SGD`) it is trained on the Udacity dog image test set.

Creating a CNN from Scratch

There are some additional questions which have to be solved when creating a CNN from scratch, beginning with the required image size. The image size for the VGG is 224x224 by definition so this question did not come up.

I chose a image size of 256x256. Since it is a little bit larger than the VGG I will not loose to much detail. By setting the size to a power of 2 (2^8) I would not have had troubles with halfing the image sizes during the pooling layers.

Second question to solve is how many convolutional and pooling layers shall the custom neural net have? The next listing 4 shows the first six layers of the VGG feature architecture from figure 4:

Listing 4: Original VGG16 Features

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): ReLU(inplace=True)
  (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (3): ReLU(inplace=True)
  (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
      ceil_mode=False)
  (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (6): ReLU(inplace=True)
  (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)
      )
  (8): ReLU(inplace=True)
  (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
      ceil_mode=False)
...
)
```

Taking a closer look at the initial stack:

1. A convolutional layer receiving 3 features (RGB values of the input image) and providing 64 output layers.
2. The second layer is also a convolutional one, providing the same amount of features 64.
3. a max pooling layer, halfing the image size to 112.
4. a convolutional layer expanding the amount of features from 64 to 128.
5. a second convolution layer.
6. another max pooling layer, halfing the image size to 56.

it can bee seen, that the amount of features roughly doubles after each pooling and the kernel size is 3 for all convolutional layers. I decided to adapt that approach and started with only one convolutional layer with 16 features.This was already able to identify 3 % of the dog images correctly, which is more it could have achieved by randomly guessing.

Afterwards the amount of features was doubled with each convolutionl layer and after each layer the dimensions where reduced by a factor of 2. The neural net with 3 convolutional layers (listing 5) was able to identify 10 % of the dog breed after training for 15 Epochs.

Listing 5: Custom Design CNN

```
class Net(nn.Module):
  def __init__(self):
    super(Net, self).__init__()
    # convolutional layer (sees 256x256x3 image tensor)
    self.conv1 = nn.Conv2d(3, 16, 3, padding=1)

    # convolutional layer (sees 128x128x3 tensor)
    self.conv2 = nn.Conv2d(16, 32, 3, padding=1)

    # convolutional layer (sees 64x64x3 tensor)
    self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
...
```

It is worthwhile to notice that the custom CNN has 64 features, but can identify 133 dog breeds. That is, because each possible combination of theses features is a different point in $\mathbb{Q}^{64\times64}$.

Refinement

- The dog images dataset could be refined using the human face detection algorithm to clean it from unwanted people in the dataset.
- Different augmentations, for example rotation of the images would lead to a better trained model, but also higher CPU/GPU usage.
- A weight initialization of the CNN would speed up the training effect and increase the prediction accuracy.

Results

Model Evaluation and Validation

- The CNN implemented from Scratch provided a test accuracy of : 10 % (88/836)
- The Transfer Learning VGG16 model provides a test accuracy of 84 % (705/836)

Justification

The Both models performed way better than I expected, I am a little proud that I was able to learn to create and adapt these models.

Appendix

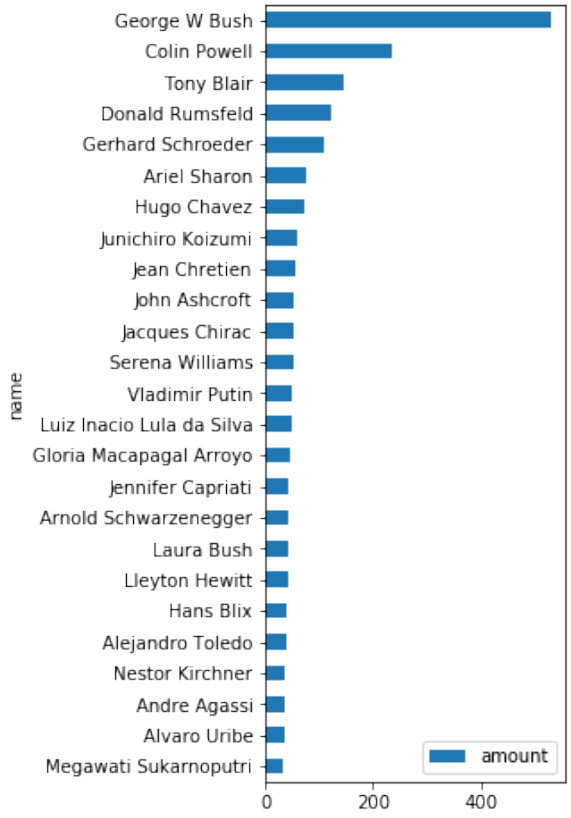


Figure 5: Top 25 celebrities represented in the human dataset

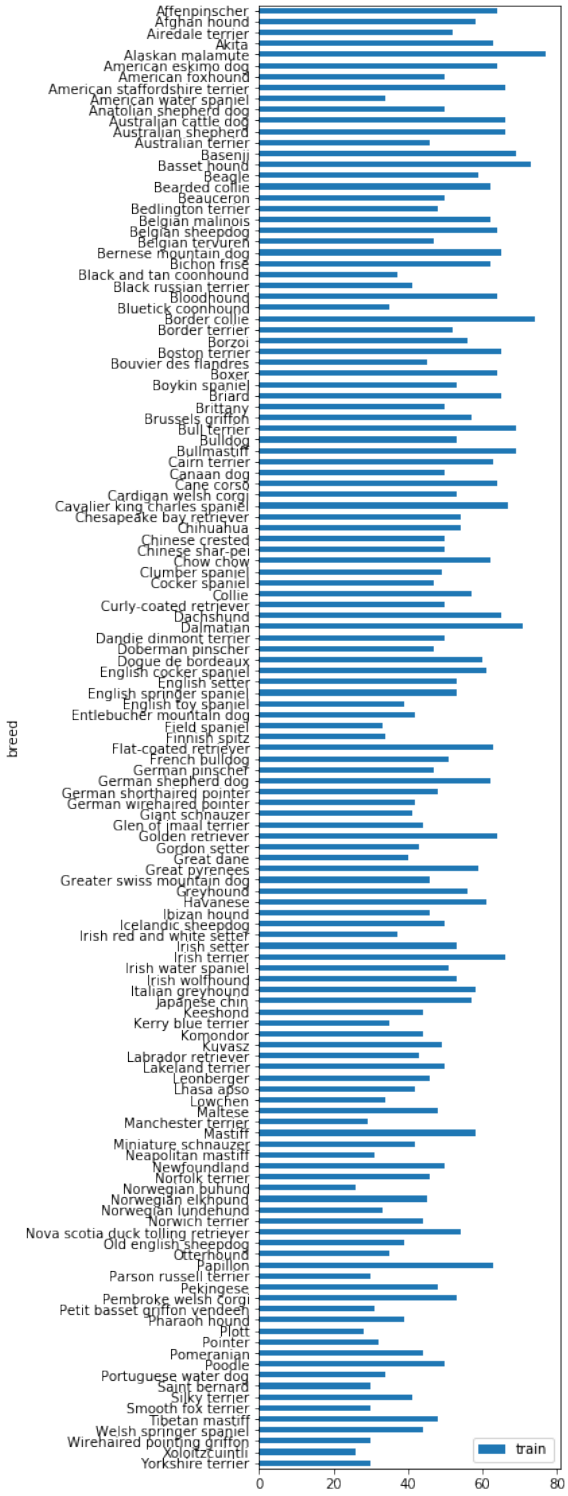


Figure 6: Balance of the Dog Breeds within the dog data set