Udacity Machine Learning Engineer Nanodegree: Capstone Project

Convolutional Neural Networks (CNN) to detect dog breeds

Matthias Hennig, March 2021

1. Definition

Project Overview

In this project I designed and trained a Convolutional Neural Network (CNN) algorithm for image recognition of dogs. Given an image of a dog, the algorithm will identify an estimate of the canine's breed. If supplied an image of a human, the code will identify the resembling dog breed.

I also used pre-trained networks such as Haar feature-based cascade classifiers from OpenCV for human face recognition and the VGG16 model developed by the University of Oxford and adapted them for the use in this project.

The end result is a model that can be deployed in a web app that takes an image as input and (1) comments whether it identified a human, dog, or neither of the two in the picture and (2) predicts the breed that the dog (OR human) resembles the most. The model has an accuracy of over 70% when performing on a test sample which can still be improved further.

Problem Statement

The goal of the project is to build a machine learning model that can be used within a web app to process real-world, user-supplied images. The algorithm has to perform two tasks:

- Dog detector: Given an image of a dog, the algorithm will provide a prediction of the dog breed.
- Human face detector: If supplied an image of a human, the code will identify the resembling dog breed.

While identifying a human or dog is basically a binary classification problem, providing a prediction of a dog breed is essentially a multiclass classification problem. In the end, our model should be able to distinguish 133 different dog breeds.

A neural network that can identify human faces or dog breeds in an image can be used to solve the problem. For this, we design and train neural networks from scratch or use pre-trained algorithms and adapt them to our use case.

Metrics

The image data will be split into a train and a test dataset. The algorithm will be trained on the former and its performance evaluated on the latter. More specifically, we can look at the model's *Accuracy*, i.e. the percentage of predictions our model got right, to evaluate how well it is able to perform the task. The goal will be the maximize the model accuracy.

In addition, part of the test dataset will be used to validate model performance during training to avoid overfitting and select the best model parametrization. This is done by using the *Cross Entropy Loss* function which is commonly used in classification tasks. In a nutshell, this function transforms the "raw" output value of the neural network to a probability for each class (using a Softmax function) and calculates the natural log of the prediction error score. The goal will be the minimize the error score.

2. Analysis

Data Exploration

The data used comprises 13,233 images of humans and 8,351 images of dogs and were provided by Udacity.

The dog dataset is split into a train (80% of images), test (10%), and validation set (10%) each containing 133 subfolders for a respective dog breed. The images all have different sizes, widths, and heights. Angles, positioning, and background vary as well.

Figure 1 Sample dog images

Breeds (Classes) are largely balanced within the sets with some classes being more present than others. For some reason, there were no images of Mastiffs provided, so it can be expected that the algorithm will perform particularly badly on this breed.

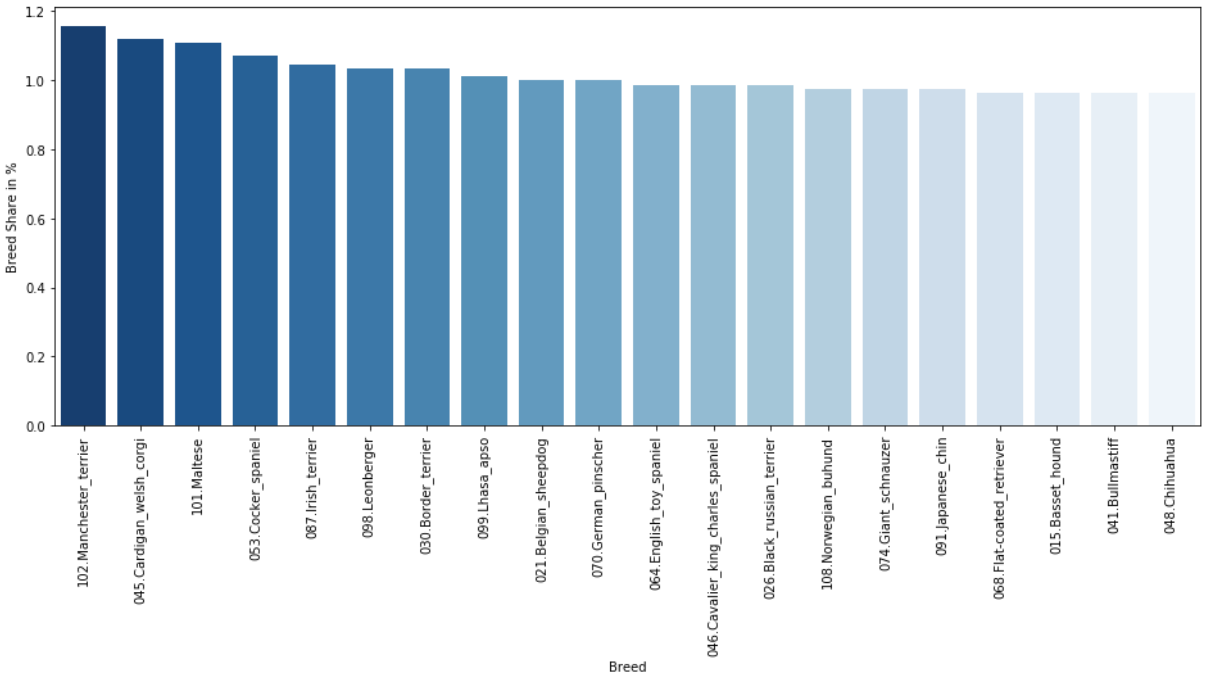

Figure 2: Top 20 Breeds by Share in Dataset

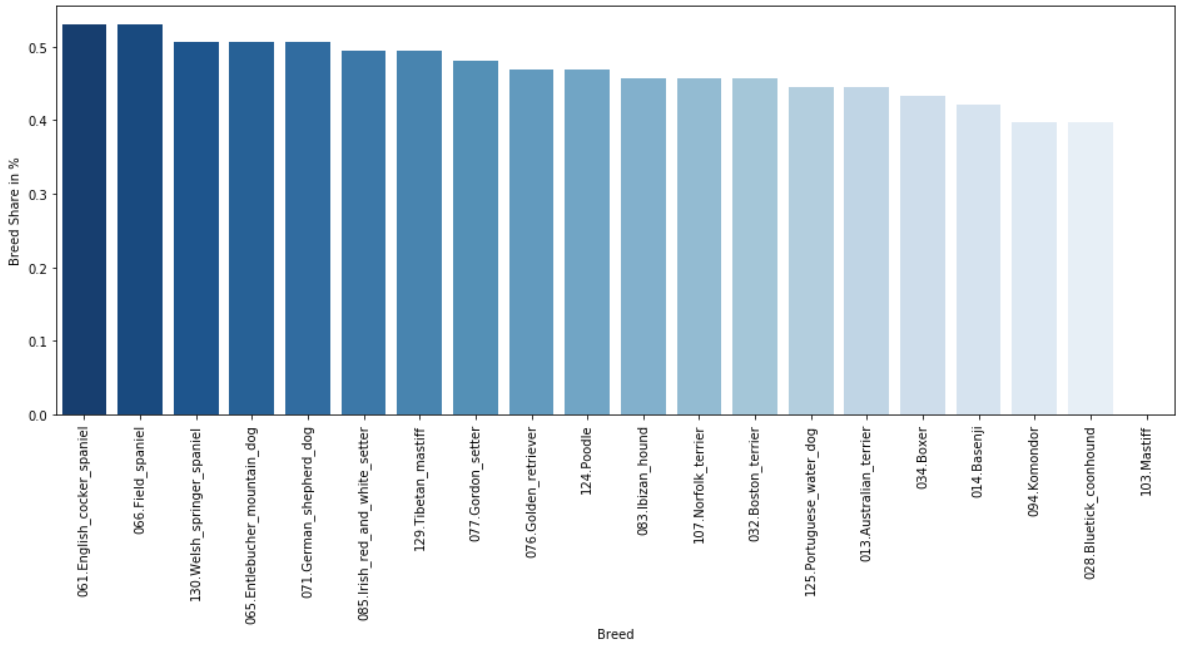

Figure 3 Bottom 20 Breeds by Share in Dataset

The human images dataset contains +5,000 subfolders for specific person, some of which contain only a single image and some multiple pictures of a person. Human files are uniformly sized (250x250) but vary by angle and background too.

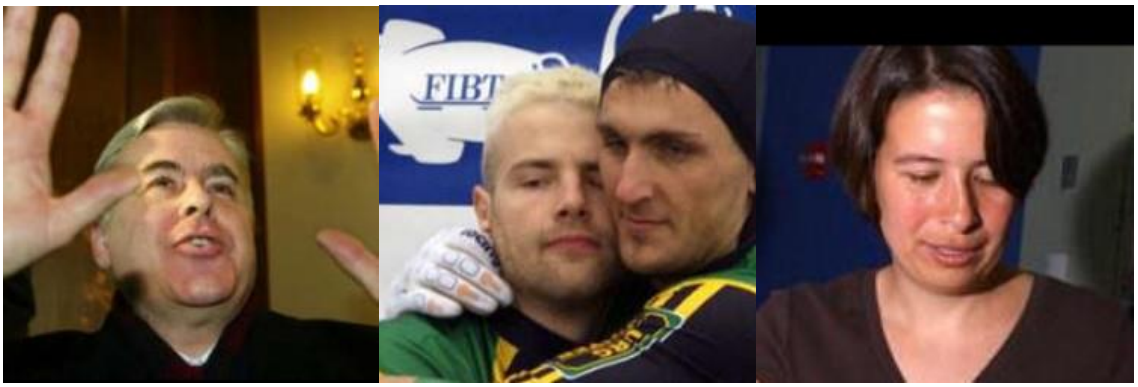

Figure 4 Sample human images

Algorithms and Techniques

Convolutional neural networks (CNN) perform very well for multiclass classification problems in general and image recognition in specific. CNNs are deep learning algorithms which can take in an input image, assign importance (learnable weights and biases) to various aspects of the image and differentiate objects in pictures. They usually consist of multiple layers that follow a certain hierarchy: First layers detect basic features like edges and pass this as an input to following layers that detect more complex features like shapes, texture, or colors. In essence, this structure allows CNNs to identify objects more effectively and efficiently. (TechTalks, 2020)

To identify human faces, I employed Haar feature-based cascade classifiers from the OpenCV library. This is a machine learning-based approach where a cascade function is trained from images containing a human face and such that don't. Shapes like edges, lines, rectangles – so called Haar features – are extracted from the images to identify eyes, nose, ears that ultimately allow the algorithm to identify human faces. (OpenCV, 2021)

For dog recognition, I firstly designed and trained a CNN by scratch and, in addition, afterwards adapted a pre-trained algorithm for enhanced model performance. I used the VGG16 CNN that achieves 93% top-5 test accuracy in ImageNet (dataset of over 14 million images belonging to 1000 classes) and has repeatedly proven to be a superior algorithm for image detection. (Neurohive, 2018)

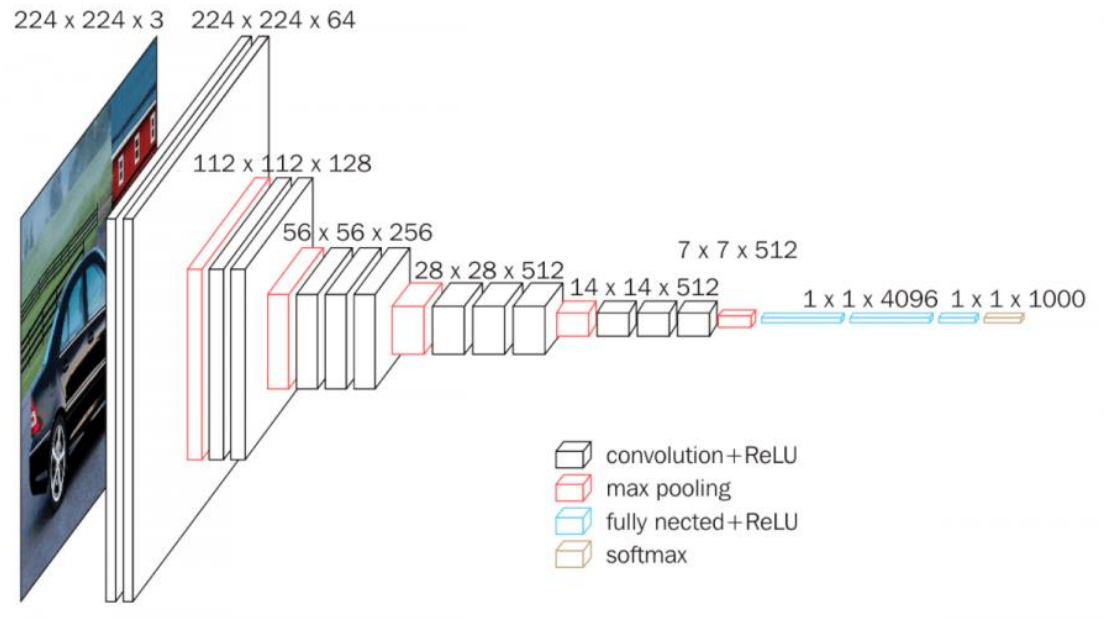

Figure 5 VGG16 network architecture

Benchmark

To pass the project, the self-trained CNN model from scratch needs to show an accuracy of at least 10%. Although not significantly reliable, this would still yield better performance than randomly guessing 1 out of 133 different breeds.

For the adapted pre-trained model, I aim for accuracy of at least 70%.

3. Methodology

Data Preprocessing

Human images are converted to gray scale before being loaded into the cascade function, as is standard procedure. No further transformations are applied.

Dog images are transformed depending on the dataset:

- Train images are resized to 258x258, augmented, center cropped to 224x224 and finally normalized after converting to tensor. Random flips and rotation prevent overfitting and best case yield better prediction performance.

- Test and validation images are only resized and cropped.

I chose a tensor size of (224,224) which is common for many pre-trained models like VGG16.

Implementation

Step 1: Human Face Detector

I use OpenCV's Haar Cascade Classifier for the human face detector. Udacity provided the "haarcascade_frontalface_alt.xml" file which is loaded into the classifier object. The detector is a function that takes an image path as input. The image is converted to grayscale. The function returns True value if one or multiple faces were recognized, otherwise it returns False value.

Step 2: Dog Detector

The dog detector is based on a pre-trained VGG16 model which comes as one of many models in the PyTorch library. The detector calls a function that takes an image as input, transforms it by cropping to (224, 224) and converting to a tensor, and applies the VGG16 model to return a number of classes. The function returns the class label with the highest score. If the label or class index is between 151 and 268 it means that a dog breed has been identified and a True value is returned, otherwise a False value is returned.[1]

Step 3 (a): Dog Breed Classifier from Scratch

The first dog breed classifier is designed and trained from scratch. My final model has **three convolutional layers**, with a kernel size of 3, stride and padding of 1. The first layer takes in an input of channel size 3, the final layer outputs a size of 256. Each convolutional layer output is ReLU-activated with a pooling layer of dimension (2,2) between each convolutional layer for better performance. The network ends in 2 fully-connected linear layers and a dropout layer at 40% before each of them to avoid overfitting. The output of both layers is **ReLU-activated**, the final output size is 133 for the number of different dog breeds in the dataset.

```
print(model_scratch)
```

```
Net(
  (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=200704, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=133, bias=True)
  (drop): Dropout(p=0.4)
)
```

Figure 6 CNN architecture from scratch

The model is trained using the **Cross Entropy Loss** function. The **Adam optimizer algorithm** with learning rate 0.05% updates the model parameters based on the computed gradients.[2]

The data is loaded to the CNN with a **batch size of 50** images and transformed as described in chapter 3, ## Data Preprocessing.

[1] See https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a for all classes on ImageNet.
[2] See (Kingma & Lei Ba, 2015).

The CNN is trained over **15 epochs**. After each epoch the error score for the validation sample is computed and each new model iteration is stored as long as the validation error keeps declining, meaning the model improves without running into overfitting risk. My final model stopped improving after 7 epochs.

Step 3 (b): Dog Breed Classifier using Transfer Learning

Transfer learning is used to seriously improve the model performance. I used the pre-trained VGG16 model and adapted the final fully-connected layer from output size 1000 to 133, corresponding to the number of different breeds in the dataset.

```
print(model_transfer)
```

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace)
    (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)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace)
    (2): Dropout(p=0.5)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace)
    (5): Dropout(p=0.5)
    (6): Linear(in_features=4096, out_features=133, bias=True)
  )
)
```

Figure 7 Adapted VGG16 architecture

Data loading and training parametrization are identical to Step 3 (a).

Step 4: Combine Detectors into Final Detection App

The final application combines all detectors. The VGG16 based model is used for breed classification due to higher accuracy.

The final app 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, returns the predicted breed.
- if a human is detected in the image, returns the resembling dog breed.
- if neither is detected in the image, provide output that indicates an error.

Refinement

My final model from scratch was obtained by trying a number of different combinations regarding model architecture (e.g. trying more/less layers, adding pooling layers, adding more fully-connected layers), data loading (different batch sizes and transformation steps), and training parametrization (number of epochs, dropout percentage, optimizer learning rate).

I used transfer learning as described in Step 3 (b) of ## Implementation to seriously improve the dog breed classifier performance.

4. Results

Model Evaluation and Validation

Human Face Detector

In a sample test of 100 images each, the model detects **98% of faces in human images** and 17% in dog images.

Dog Detector

In a sample test of 100 images each, the model detects **75% of dogs in dog images** and 1% in human images.

Dog Breed Classifier from Scratch

Evaluating the final model on the test dataset yielded an **accuracy of 13%**, i.e. correctly classifies 108 out of 836 dog breeds.

Dog Breed Classifier using Transfer Learning

The VGG16 based model yielded an **accuracy of 73%** on the test dataset, i.e. correctly classifies 627 out of 836 dog breeds.

Final Detection App

On a self-provided sample of 8 images (3 human images, 5 dog images) - that the models haven't seen before – the final app correctly identifies the humans and dogs. In contrast to that, dog breeds identification showed room for improvement. Two pictures of German shepherds have been misclassified as Pomeranian and Anatolian shepherd dog.

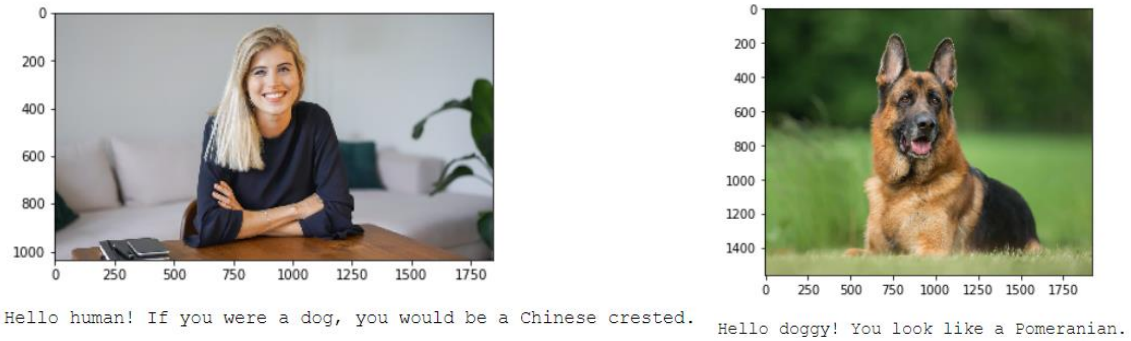

Figure 8 Sample output of final detection app

Justification

The human and dog detectors work well enough to provide an adequate solution for our problem. The CNN to classify dog breeds that I have written from scratch meets the 10% accuracy benchmark. However, it's still not good enough to be reliably applied in the final application. Using the pre-trained VGG16 yields much better performance and so is applied in the detector app.

But even with this algorithm in place, dog breed classification performance shows still much room for improvement. Possible improvements could be made by:

1. Adding more layers to the self-designed CNN or using different pre-trained models such as ResNet-50 that features 50 layers or Google's Inception V3.
2. Increasing the number of epochs to train the models. However, in my case, more training iterations wouldn't have helped as the validation score did not decrease further.
3. More image datasets of dogs and specific dog breeds. As the provided dataset is slightly imbalanced a particular breed might be underrepresented and therefore harder to correctly classify as other breeds.
4. More image augmentations (flipping vertically, move left or right, etc.) could improve overall performance on test data.

5. References

Kingma, D. P., & Lei Ba, J. (2015). *ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION.* Von https://arxiv.org/pdf/1412.6980.pdf abgerufen

Neurohive. (2018). *VGG16 – Convolutional Network for Classification and Detection*. Von https://neurohive.io/en/popular-networks/vgg16/ abgerufen

OpenCV. (2021). *OpenCV Cascade Classifier*. Retrieved from https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html

TechTalks. (6. Januar 2020). *What are convolutional neural networks (CNN)?* Von https://bdtechtalks.com/2020/01/06/convolutional-neural-networks-cnn-convnets/ abgerufen