Dog breed classification

I. Project Definition

One of, if not the most important, concept of a data science is the classification algorithm. You can predict the target class by analyzing the training dataset and optimize and optimize the whole thing to the desired result.

The Udacitiy Capstone project dog breed classification is exactly that. The goal of the project is to determine the dog breed on a picture if this picture contains a human or a dog. If the picture shows neither, a message with this information should also be the output.

II. Strategy

Due to the requirements, the project can be divided into serval steps

- Step I: Data analysis
- Step II: Human face detector – Recognition function for human faces
- Step III: Dog detector – Recognition function for dogs
- Step IV: Creating a Connected nural network to cassifiy dog breeds
- Step V: Dog breed predictor – function that combines the dog and human detector and provide a breed prediction.

For this purpose Udacity provided a Juptyer notebook as guideline

III. Approach

Step I: Data analysis

Also the following datasets have been provided:

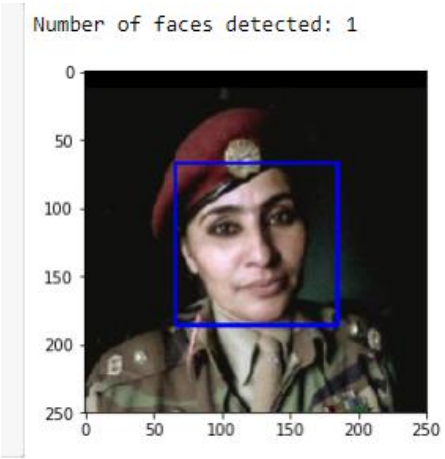
```
There are 133 total dog categories.
There are 8351 total dog images.

There are 6680 training dog images.
There are 835 validation dog images.
There are 836 test dog images.

There are 13233 total human images.
```

Step II: Human detector

With the help of the provided notebook, the function for recognizing human faces, should now be written. For this purpose, a pre-trained face detector called Haar feature-based cascade classifier was used. This managed to recognize a face in 100% human pictures, but also in 11% of the dog pictures.




```
import cv2
import matplotlib.pyplot as plt
%matplotlib inline

# extract pre-trained face detector
face_cascade = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_alt.xml')

# load color (BGR) image
img = cv2.imread(human_files[3])
# convert BGR image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# find faces in image
faces = face_cascade.detectMultiScale(gray)

# print number of faces detected in the image
print('Number of faces detected:', len(faces))

# get bounding box for each detected face
for (x,y,w,h) in faces:
    # add bounding box to color image
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)

# convert BGR image to RGB for plotting
cv_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# display the image, along with bounding box
plt.imshow(cv_rgb)
plt.show()
```

Step III: Dog detector

With the help of the pre-trained model ResNet-50, it was possible to develop a dog detector. That detector can recognize a dog in 100% of the time if there was one in the picture. In addition, no dog was recognized in the pictures in which only a human was in.

This enables us to clearly identify a human face and a dog with the help of the two detectors. Vice versa, all images in which neither a dog nor a human can be recognized as well.

Step IV: Dog Creating a Connected nural network to cassifiy dog breeds

Based on the task, the first thing to build was a CNN from scratch, which should have an accuracy of minimum 1% without using transfer learning. My model had this structure:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 16)      1216
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 112, 112, 16)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 32)      12832
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 56, 56, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 56, 56, 64)        51264
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 28, 28, 64)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 50176)             0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               12845312
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 133)               34181
=================================================================
Total params: 12,944,805
Trainable params: 12,944,805
Non-trainable params: 0
_________________________________________________________________
```

More details about the model can be found in the Jupyter notebook:

dog_app_final.ipynb

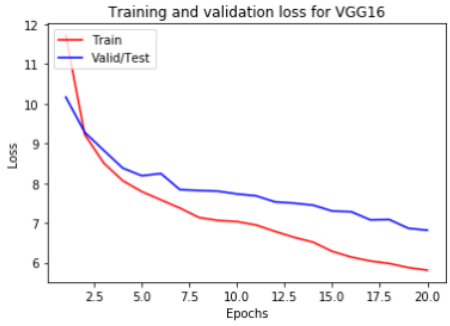
Applying transfer learning

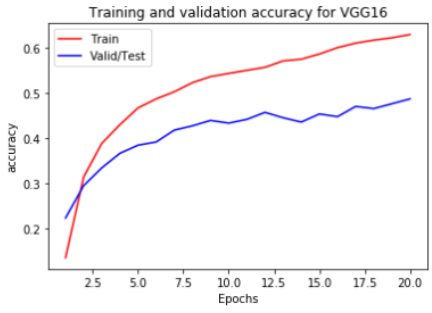
Building a model by hand that can reliably predict the breed of dog would take forever, this is where transfer learning comes in.

We are loading a pre-training CNN that has already been trained on large data sets. Freeze the weighting, add individual layers and parameters that are necessary for our needs and can thus achieve a very good result in a shorter time.

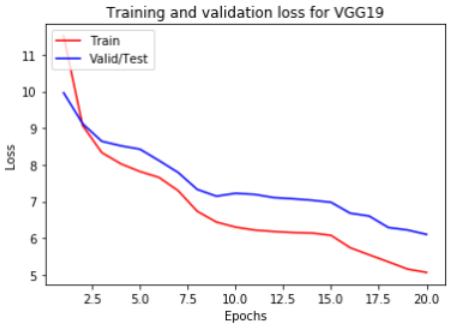
The following pre-training CNN came into question and were tested in 20 epochs:

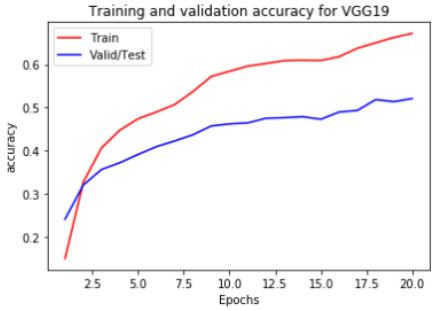
VGG-16
Test accuracy: 49.64%



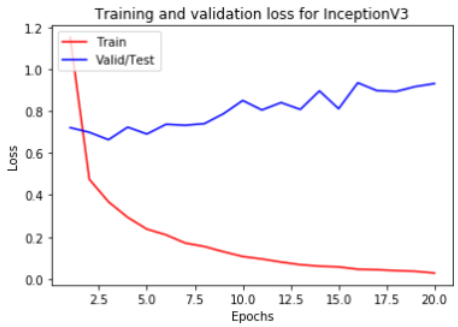


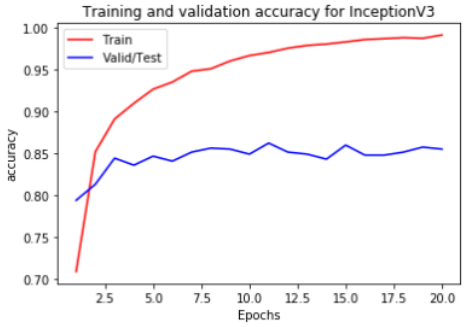
VGG-19
Test accuracy: 52.51%



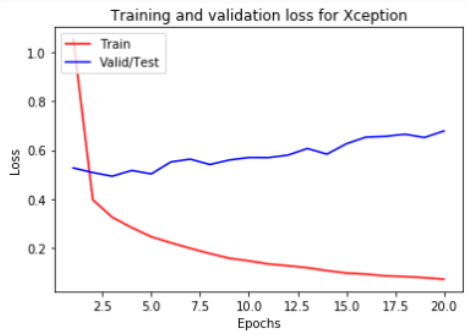


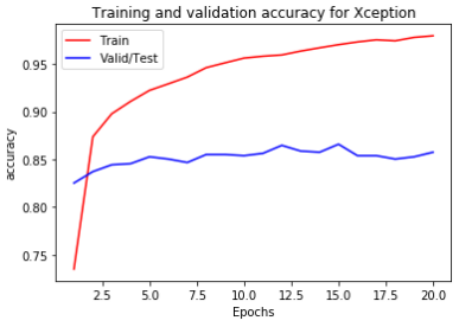
Inception V3
Test accuracy: 77.99%



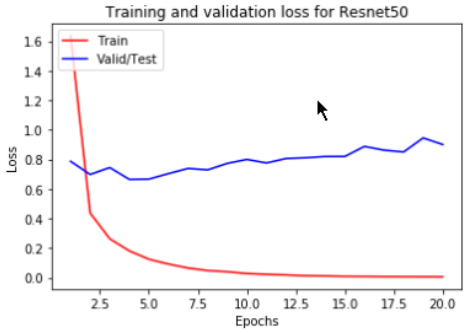


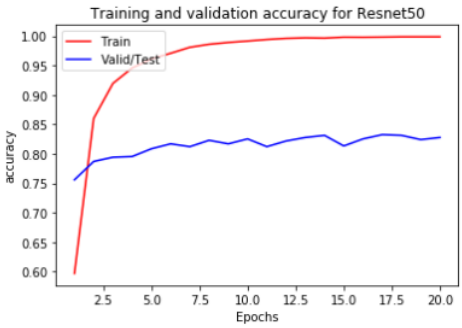
Xception
Test accuracy: 83.85%





ResNet 50
Test accuracy: 79.90%





Xception showed the highest accuracy in the test and VGG-16 the lowest.

After testing the networks, I decided to use Resnet 50. While Xeption supplies a slightly better test accuracy the size of the bottleneck features is too unwieldy for me. ResNet as a pre-trained network delivers a similarly good performance with almost 80% test accuracy.

After a few small adjustments, the following model resulted:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_1 ( (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               524544
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 133)               34181
=================================================================
Total params: 558,725
Trainable params: 558,725
Non-trainable params: 0
```

With a Test accuracy of: 80.86%

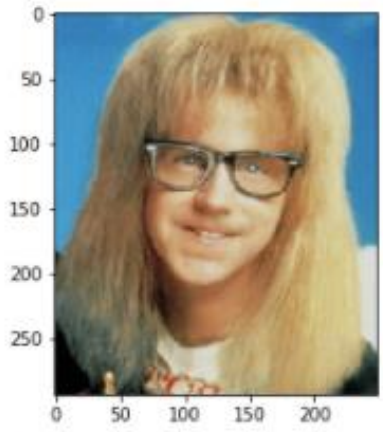
Step V Dog breed predictor

Finally, the two detectors were combined to a dog breed predictor.

The first step in this is to check whether it is a dog. If so, it will be confirmed in your text and the respective breed will be determined.


```
This is a dog and it's breed is Labrador retriever.
```

If the Dog Dector does not recognize a dog, the image if it contains a human face. If so, most similar dog breed is determined.



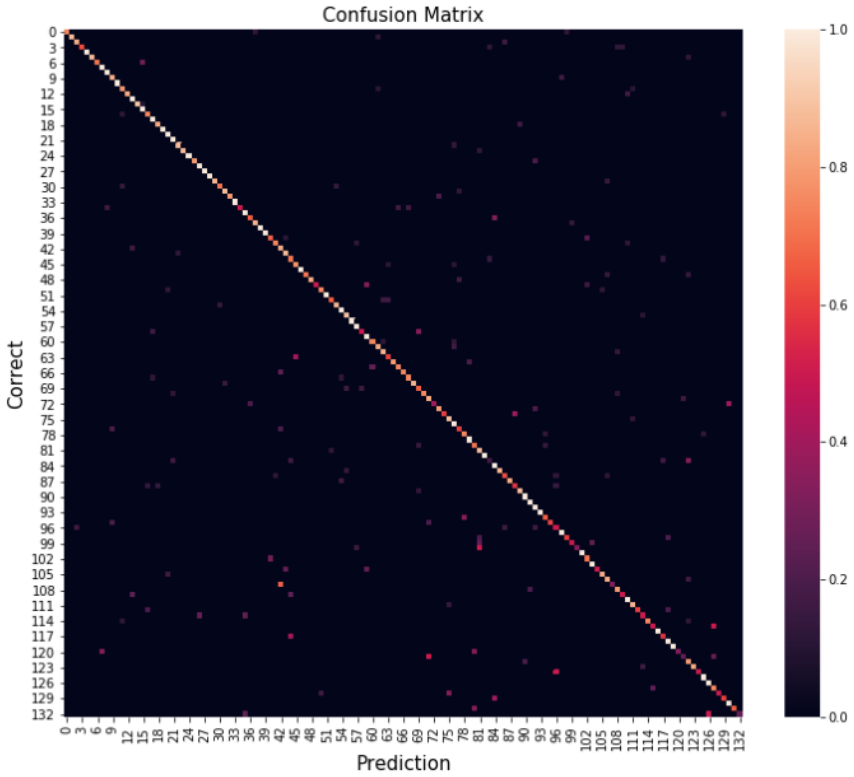

```
This is not a dog but it's look like a Cavalier king charles spaniel.
```

If neither applies to both, this is output with a note.

```
This looks niether human or dog, must be something else.
```

IV. Conclusion

If the result of the model is viewed as a confusion matrix, it looks like this:



A large part of the dog breeds is recognized perfectly.

Individual outliers are closely related breeds of dogs that are very similar, the respective picture depends on which breed is determined.

From my point of view it works very well, the algorithm only has its difficulties with similar breeds. These could be remedied with the following adjustments.

Architecture:

- Increase the model capacitiy by adding more layers
- change the batchsize or change the learing rate
- increase the number of epochs

Redesign the image:

- change the resolution
- rotation or flipping the image
- random image shifts

For further details feel free to dive directly in to the notebook!