Project Report – Machine Learning Engineer Nanodegree
Laura Hartzheim, 21.12.2020
github:
https://github.com/Laura9944/capstoneProject

Dog Breed Classifier (CNN)

1. Definition

Project Overview

Image Classification is a common supervised learning task. In this project images will be classified into dog breeds. The goal is that a user will upload a picture and the algorithm will then predict a dog breed to match the image. This will be done using a convolutional neural net.

Problem Statement

The goal of this project is to create an algorithm, that allows users to use their pictures for
classification. If the image contains a human or a dog, a dog breed will be predicted, if not
an error message should be displayed. In addition, the algorithm will give notice if a human is actually in the image, but will still predict a dog breed.
To achieve this three different neural nets will be used. One to detect if an image contains a dog, one to detect a human face and the dog breed classifier.

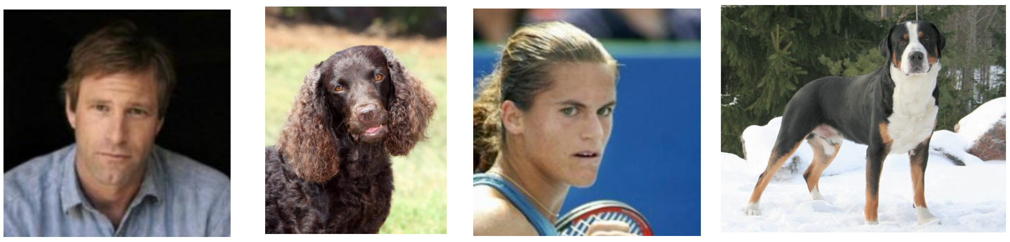
Metrics

The models' performance will be measured based on accuracy. This is because there are specified goals based on accuracy in this project. To meet these goals a benchmark model will be created from scratch. This model will have to achieve an accuracy of at least 10% when predicting dog breeds. After the benchmark model has been created the actual classifier will be created using transfer learning. This model will have to achieve an accuracy of at least 60%. The accuracy is calculated by dividing the number of correct classifications by the number of total classifications. It allows for an easy and comprehandable comparison between the two models. To deal with the imbalance of the dataset I added F1 as an addditional metric to the transfer learning net.

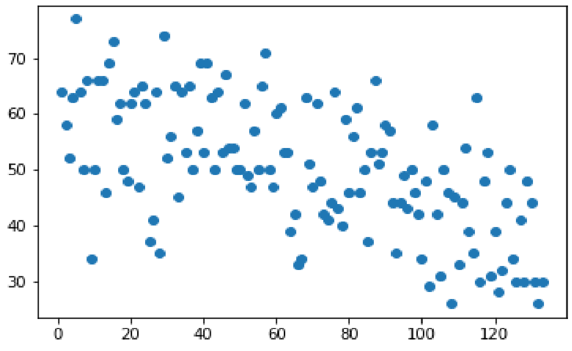
2. Analysis

Data Exploration

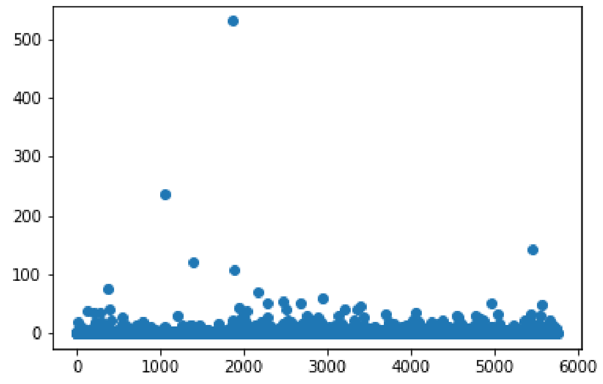
The dataset is provided by Udacity. It can be split into two parts. One part contains pictures of humans. It is made up of 13233 photos, that are grouped by the names of the people that can be seen in the images.
The second part of the dataset contains images of dogs. There are 8351 pictures included. These are labeled with the breed of the dog that can be seen in the image. There are 133 classes represented by this dataset. The dog breed data already comes split into training, test and validation set. The images in this dataset are not consistent in sizing.
The number of images per class is not constant in the human and the dog dataset. This means that both datasets are imbalanced.



For the dog dataset this can bee seen in the plot below. The x-Axis shows the labels(dog breeds). The y-Axis shows the amount of images in the training set for each class.



The plot oft the human dataset also shows some irregularities in the amount of images per class. This can be seen below.



Algorithms and Techniques

In the following I am going to describe what Algorithms and Techniques have been used for each step. The whole project is implemented in a jupyter notebook using pytorch.

<u>Step 1: Detect Humans</u>

To detect whether an image contains a human face a pretrained implementation of a haar feature-based cascade classifier was used. To use this the images need to be transformed to grayscale. The classifier will return a bounding box a round the face. In this case we only need to know if the picture contains a human face. The output can be transformed to a boolean by checking if the length of the output is bigger than zero. This means that a face was detected(-> True).

<u>Step 2: Detect Dogs</u>

To detect if an image contains a dog a pretrained version of VGG-16 will be used. The model has been trained with ImageNet data, which has about 1000 classes. To use this model images need to be cropped to 244x244. The output will then be turned into a boolean by checking if it is between 150 and 269, since these are the dog related classes.

<u>Step 3: CNN to Classify Dog Breeds from Scratch</u>

To create this classifier the images from the dog dataset will be used. The images will be cropped to 244x244. The architecture of the net is inspired by VGG-16. However, it is a lot smaller, to allow it to run on standard laptops in a reasonable time. The exact architechture can be seen in the section „implementation“. Cross-Entropy Loss is used, because the problem contains more than two classes.

Step 4: CNN to Classify Dog Breeds using Transfer Learning

The same dataset with the same transforms and the same loss function will be used. Transfer learning means that only a part of an already trained net will be replaced and retrained. This is done to adapt net that is known to perform well to a new usecase. The net used for transfer learning is a pretrained resnet50. Only the last layer is changed. This layer is linear, the amount of its output neurons is changed to 133.

Step 5: Write the Algorithm (Combine previous steps)

In this step all previous steps will be combined. The first step is to use the dog detector if the image contains a dog. If so the dog breed will be detected using one of the CNNs. If no dog is detected, the human face detector will be used. If this returns true the dog breed CNN will be used again. If no human and no dog could be detected an error message will be displayed.

Step 6: Testing the Algorithm

In this final step the algorithm is tested using two images each of a human, a dog and something else. These images are not part of the datasets described above.

Benchmark

The CNN created from scratch will serve as the benchmark model. It has to achieve an accuracy of at least 10%, this goal is set by udacity. The transfer learning model will have to compete angainst it, meaning it will have to achive better results.

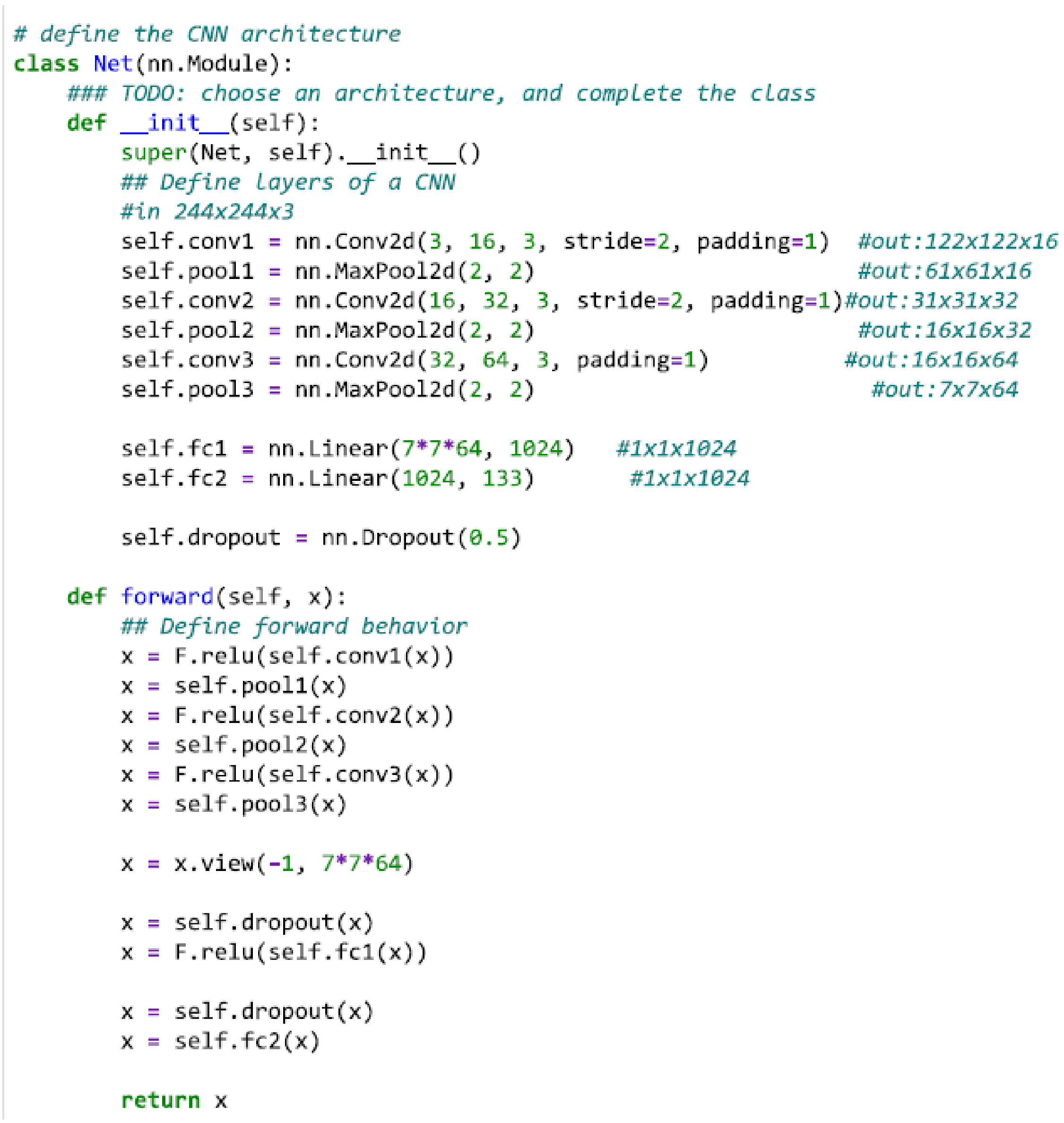
3. Methodology

Data Preprocessing

The data is cropped to 244x244 in all cases. This shape is required by the VGG-16 net. If a net contains parts that are not convolutions it is necessary to have a consistent input size. That is why the 244x244 size is used for all nets in this project. There are no other augmentation used in this project. However, they could be used to further improve the results.

Implementation

In this part I am going to discuss the implementation of the models used in this project. The CNN from scratch is inspired by the VGG-16. The input for this net is 244x244x3. The first layer is a 3x3 convolutional layer with stride two and padding one. It has 16 output channels. This is followed by ReLU. Then a 2x2 max pooling layer with stride two is used. This structure is iterated three times. The last layers of the net are two fully connected layers, each proceeded is by dropout.



```python
# define the CNN architecture
class Net(nn.Module):
    ### TODO: choose an architecture, and complete the class
    def __init__(self):
        super(Net, self).__init__()
        ## Define layers of a CNN
        #in 244x244x3
        self.conv1 = nn.Conv2d(3, 16, 3, stride=2, padding=1)  #out:122x122x16
        self.pool1 = nn.MaxPool2d(2, 2)                        #out:61x61x16
        self.conv2 = nn.Conv2d(16, 32, 3, stride=2, padding=1)#out:31x31x32
        self.pool2 = nn.MaxPool2d(2, 2)                        #out:16x16x32
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)          #out:16x16x64
        self.pool3 = nn.MaxPool2d(2, 2)                         #out:7x7x64

        self.fc1 = nn.Linear(7*7*64, 1024)   #1x1x1024
        self.fc2 = nn.Linear(1024, 133)       #1x1x1024

        self.dropout = nn.Dropout(0.5)

    def forward(self, x):
        ## Define forward behavior
        x = F.relu(self.conv1(x))
        x = self.pool1(x)
        x = F.relu(self.conv2(x))
        x = self.pool2(x)
        x = F.relu(self.conv3(x))
        x = self.pool3(x)

        x = x.view(-1, 7*7*64)

        x = self.dropout(x)
        x = F.relu(self.fc1(x))

        x = self.dropout(x)
        x = self.fc2(x)

        return x
```

Code of the CNN from Scratch

The transfer learning net is created using a pretrained ResNet50. To use this the parameters of the model had to be frozen first. Then I proceeded by replacing the fully connected part of the net with a linear layer with 2048 input neurons and 133 output neurons. The model was then retrained with the all parameters frozen except for the linear layer.

4. Results

Model Evaluation and Validation

Both CNNs were validated using the validation past of the dog dataset. The accuracy was determined using the test data.
The CNN from scratch received a test loss of 3.93 and an accuracy of 12%. This surpasses the goal of an accuracy of at least 10%.
The transfer learning CNN received a test loss of 0.548 and an accuracy of 83%. This surpasses the goal of 60% by a lot.
In the final step of the project some additional testing was done by trying the whole algorithm on six pictures that were not part of the datasets mentioned above. The algorithm showed correct behavior for the four pictures that did not include dogs. For the dog images one was classified correctly the other was identified as a dog but the breed was incorrect. However, the wrongfully predicted breed looks quite similar to the actual one.

Justification

Both convolutional neural nets surpass the required goals. The CNN from scratch is only 2 percent points better. However, the transfer learning model showed great results. It is more than 20 percent points better than required. The transfer learning net performs a lot better than the benchmark model/ CNN from scratch. The accuracy of the transfer learning model is quite high and the F1 score of over 0.8 is also a good result. The model is definetley good enough to solve the problem of detecting a dogs breed.

Possible Improvements

The accuracy of both nets could be further improved using data augmentation. This would create more images to train on. Searching for more datasets with dog breeds could also help to increase the accuracy. It would also be possible to further experiment with the architecture of the CNN from scratch.