Capstone Project

Machine Learning Engineer Nanodegree

Volodymyr Mudryi

May 30, 2020

Dog Breed classification with Convolutional Neural Networks

Definition

Project Overview

Today machine learning can solve a large number of tasks, so I decided to classify a dog breed by photo.

The result of this project is App to classify a dog breed by the photo, in case if there are humans instead of the dog on the image app will return the dog breed which is the most similar to that human.

Problem Statement

The goal is to create a pipeline with a dog detector on the photo if there any dogs a human face detector should run and after that, a CNN should take a photo and return a correct dog breed for dog's pictures and most similar dog breed for human pictures.

In this task, I implemented 2 CNN models, first one is created from scratch (used as a benchmark) and the second one is based on the transfer learning approach.

Metrics

This is a classification task with a lot of classes, so a common approach will be to use multi-class log loss metrics to model evaluation during train process:

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

Where ***M*** is number of classes, ***y*** - binary indicator (0 or 1) if class label ***c*** is the correct classification for observation ***o***, ***p*** - predicted probability observation ***o*** is of class ***c***

And to measure model results I use accuracy score:

$$accuracy = \frac{true\ positives + true\ negatives}{dataset\ size}$$

For human photos there is not any metrics because we don't have correct labels, so the only metric here is human base evaluation (ask some group of people)

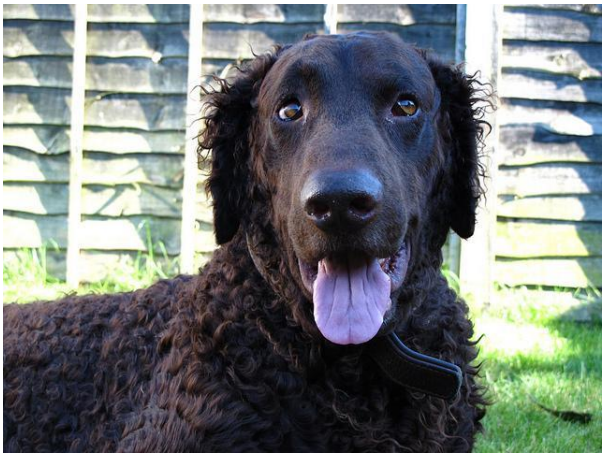
Analysis

Data Exploration

Dataset for this task was provided by Udacity, so there is a human photos folder and dog photos folder. All photos were labeled by the correct dog breed (class).

- For human pictures, we got 13233 images without any splits because we use it only for test pre-trained models and wouldn't train any human recognition algorithm. Therefore, all human pictures are of the same size (250×250)
- For dogs pictures, we got 8351 splits into three folders: train (6,680 Images), test (836 Images), and valid (835 Images) which divided by folders by dog breeds (133 folders in all splits). All these pictures have different sizes, lightning, backgrounds, and dog position so it will help to build a more robust algorithm.

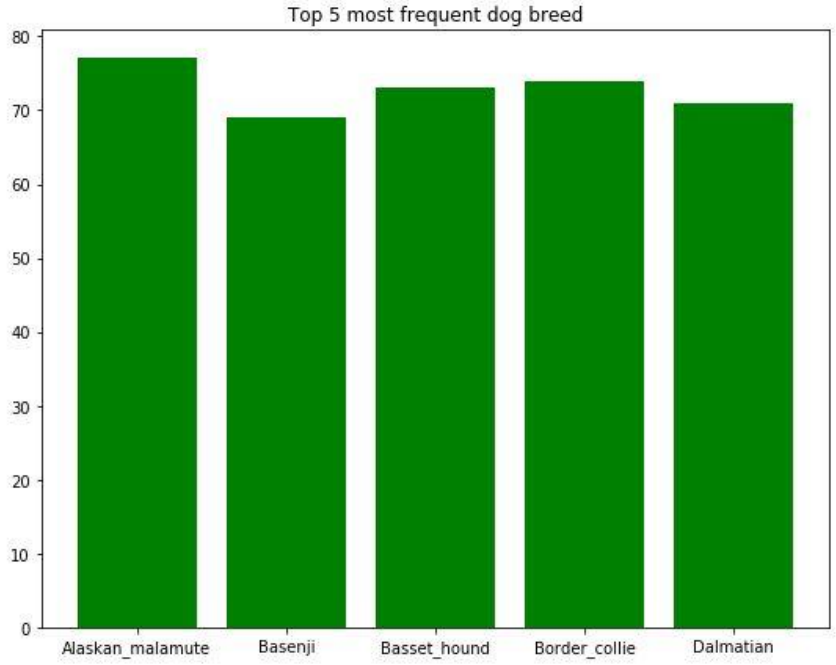
Dog pictures example:

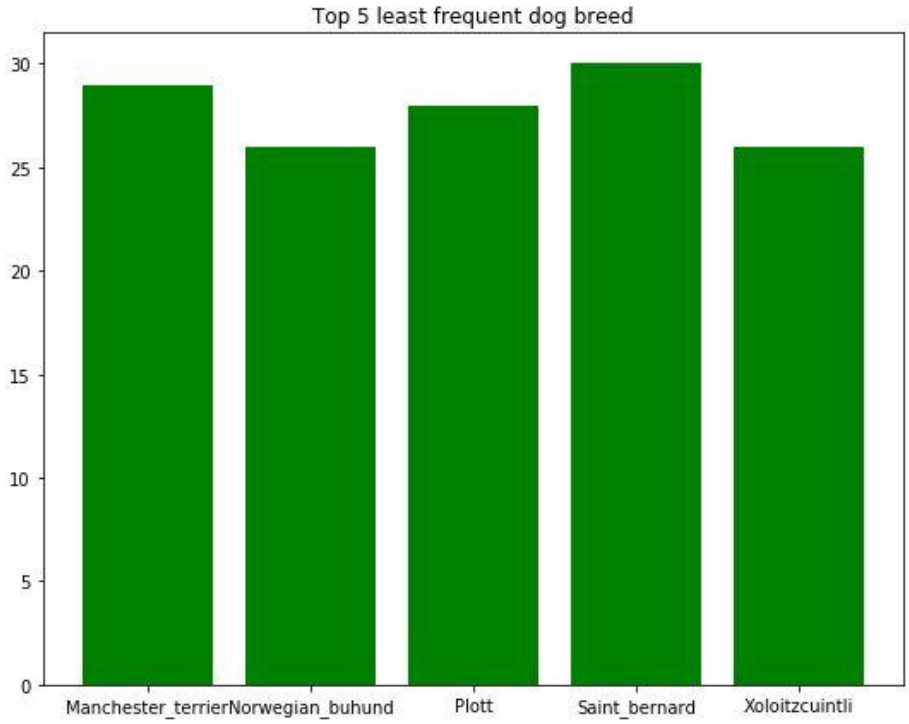


Human pictures example:

Exploratory Visualization

For this task, there aren't many possible explanatory visualizations because all the data are in photos, but I think that could be useful to look at top 5 most frequent and least frequent dog breeds in train data





As we can see there is disbalance in data because the most frequent dog breed is:

- Alaskan_malamute appears 77 times
- Border_collie appears 74 times
- Basset_hound appears 73 times
- Dalmatian appears 71 times
- Basenji appears 69 times

And the least frequent appears almost half that much:

- Norwegian_buhund appears 26 times
- Xoloitzcuintli appears 26 times
- Plott appears 28 times
- Manchester_terrier appears 29 times
- Saint_bernard appears 30 times

So it should be careful to detect all the small size classes. Because the algorithm can simply label all as a majority classes

Algorithms and Techniques

I used a CNN architecture because it's the best approach when you want to work with images because it can extract features from images very effectively. On top of it, I use a few fully connected layers for classification itself. I also add some sort of regularization with data augmentation and Dropout layers. But in fact, the CNN networks need a lot of data to train effectively so it would be possible to train it from zero with the provided amount of data.

The next step was to use a transfer learning, it provides a pre-trained CNN part (feature extracting) so only aa classifier part is training, so with this approach result should be much better.

Benchmark

The first benchmark was a random guess, even with high data imbalance this approach could give an accuracy score of less than 1%.

The next step was to provide a model from scratch and achieve at least 10% of accuracy. After that, we can use it as a benchmark.

And for final transfer learning approach we should beat a scratch model and achieve an accuracy score of more than 60%.

Methodology

Human face detector

For human face detector, I used an OpenCV approach with Haar feature-based cascade classifiers. Haar Cascade is a machine learning object detection algorithm used to identify objects in an image and based on the concept of features proposed by Paul Viola and Michael Jones in their paper "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001.

In the data preprocessing step image got grayscaled and after that, a simple CascadeClassifier detect faces on image and return number boolean variable if that any faces on the image.

Dog detector

For the dog detector, I used a pre-trained VGG-16 Neural Network Model. This model trained on ImageNet (huge image dataset).

In data preprocessing step image got resized to be 224x224 and after that normalize with following parameters: mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]. Then a model classifies the image and return a number, one of 1000 possible classes. In our task, we only need dogs, so in that case, we should label as a dog all classes that related to different dog breeds (158-268).

Dog breed classifier

Training of CNN for dog breed classification.

1. Data Preprocessing

For data preprocessing i used a following steps: resize to 256x256 (because it’s a common size for CNN models) add a RandomHorizontalFlip for data augmentation (only for train dataset) and normalize image with following parameters mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225].

2. CNN from scratch

For this task, I have a lot of experiments but in the end, I decided to recreate a famous AlexNet architecture, so I end up with these layers:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 63, 63]          23,296
         MaxPool2d-2          [-1, 64, 31, 31]               0
            Conv2d-3         [-1, 192, 29, 29]         307,392
```

```
         MaxPool2d-4          [-1, 192, 14, 14]               0
            Conv2d-5          [-1, 384, 14, 14]         663,936
            Conv2d-6          [-1, 256, 14, 14]         884,992
            Conv2d-7          [-1, 256, 14, 14]         590,080
           Dropout-8                [-1, 50176]               0
            Linear-9                 [-1, 4096]     205,524,992
          Dropout-10                 [-1, 4096]               0
           Linear-11                 [-1, 4096]      16,781,312
           Linear-12                  [-1, 133]         544,901
```

For loss I use CrossEntropyLoss and as an optimizer, I use SGD (Gradient descent (with momentum)) with a learning rate equal to 0.05 and a train model for 25 epochs with a batch size equal to 32.

3. Transfer learning

For transfer learning, I decide to use a ResNet101 pre-trained model. I choose ResNet because it's like one of the best models architecture for image classification and I often see mention of this architecture in many Kaggle competitions and articles (mainly because of skip-connections).

Also, its main improvement of this step it's not in architecture its more in parameters that trained on a large dataset. So with this pre-trained model for features extracting I change a fully connected layer on top to be suitable for my task (133 class outputs).

For loss I use CrossEntropyLoss and as an optimizer, I use SGD (Gradient descent (with momentum)) with a learning rate equal to 0.001 and a train model for 15 epochs with a batch size equal to 32.

Results

Model Evaluation and Validation

In the end, the final model achieves an accuracy score 87% what it good results, but not the best one there are still a lot of improvements but all of them it's not clearly its more like more parameters to be analyzed but in general model perform pretty good, I upload 3 dog photos and 2 of them are correctly labeled and the one did not recognize dog on it, so there should be a lot of work to improve a dog recognition algorithm to make it more robust. Also, I think with more time and more data it model can achieve better results.

Also, the model from scratch achieve 11% accuracy which is low result but for me, the main purpose was to build this model because my main goal was to learn how to develop such a models.

So, in conclusion, it thinks the model performs good enough with this amount of data (less than 60 photos per class on average) so with more data it could be improved, but the main improvement should come from the dog detector algorithm. Also with more data, we can train the model for more epochs because small dataset models easily overfit.

Justification

In the end, this model achieved CrossEntropyLoss equal to 0.68 and accuracy equal to 87% of its good result for me, but in comparison with some competition on Kaggle where the top result for loss is equal to 0 its pretty bad. But on Kaggle there was dataset 3 times bigger so in future I think I will try to use my models with that dataset.

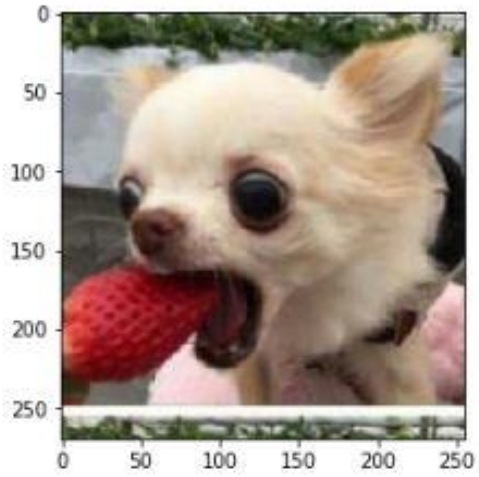
Also the part about human it's pretty fun, with human evaluation from my friends its look like everything work fine (fine enough to make some fun) so in general conclusion this project results not the best but it's good enough with such dataset and such model training environment.

Conclusion

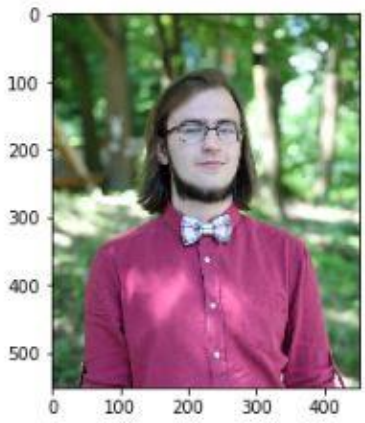
Free-Form Visualization

In this section I provide an example result of the model output:

The correct results of the algorithm:

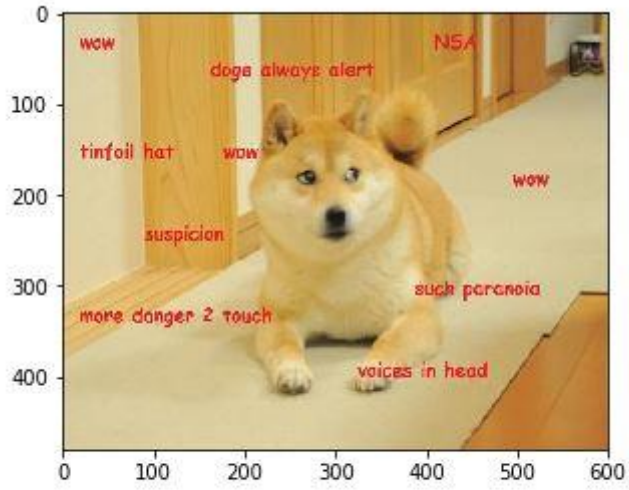


```
Dog detected.
Its look like a Chihuahua
```



```
Hello human
You looks like a English springer spaniel
```

The bad results (error in dog detector model)




```
Sorry, but any human or dog was detected
```

Reflection

For me, this project was really interesting because I learn a lot of new stuff about data loaders, can write my own implementation of famous architecture, and make it work and combine a few models to make a final result as an app. Also was interesting to tune different parameters and look at how the loss goes down or up with different sets of parameters.

For me, some difficulties were in summarize all the results, like make a GitHub repo with all the stuff, wrote this report also takes me a long time and power.

In the future, I have some thinks about how to transform this model and improve results to build a funny app where you can recognize dogs from memes and find the most similar dog photos to you.

Improvement

In general, results are pretty good, but there are some steps to improve the app pipeline:

- Add more data augmentation (rotation, blur, cropping) and more data
- Choose the best model parameters and maybe train with those parameters a bit more time
- Use better human face and dog detectors, because there was a photo where the dog is not detected

Also, I think a possible improvement of the project in general:

- Show top n dog breed for prediction, so the user can choose between 3-4 breed the best one
- For human its looks cool to show a picture of that breed so it may be implemented a similarity searching between human photo and dogs photo of that breed.