Magali Semeria
Udacity ML Engineer nano degree
2021-02-14

Capstone Project Report

Dog breed classification web application

Image from https://www.bbc.co.uk/newsround/48673084

Hello dog! I think you are a ... Boykin spaniel

Definition

Project Overview

In this project, we build a web application that classifies dog breeds. Much of this section's content is already detailed in the document « proposal.pdf ». This project illustrates a task often encountered in machine learning: image classification. Image classification is used in various domains. For instance, a famous application of image classification identifies hand-written digits[1].

We build a simple web application that takes as input a picture of a dog and returns the name of the corresponding dog breed. As suggested in the project notebook, we first develop a **dog detector** that will return True if a dog is present in the uploaded picture. We then develop a **dog breed classifier** that will return the dog breed of the dog in the image. For additional fun,

[1] https://ieeexplore.ieee.org/abstract/document/576879/

we develop a **human face detector** that will detect if a human face is present in the uploaded image. If so, we use the dog breed classifier to return the predicted dog breed for the human. If neither human or dog is detected in the picture, the application does not return a prediction.

The application's workflow is represented below:

User uploads a picture

Image is pre-processed to fit models' input format

Dog detector predicts if the image contains a dog

No

Yes

Human face detector predicts if the image contains a human

No

Yes

The application does not display a prediction

Result

Hello dog! I think you are a ... Bulldog

Dog breed classifier predicts a dog breed

This project includes two main stages:

1. Model development: we develop the models and algorithms behind the dog detector, the human face detector and the dog classifier. For this stage, we follow the instructions included in Udacity's dog classification project notebook[2]. We use Udacity's dog dataset available at https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip to train and test the dog breed classification models. We use the University of Massachusetts « Labeled Faces in the Wild »[3] dataset available at http://vis-www.cs.umass.edu/lfw/lfw.tgz to assess the performance of the human detector model.
2. Web application development: we integrate the dog detector, human face detector and the dog classifier into a simple web application using Flask[4].

Problem Statement

Dog breed classification is a great candidate for a transfer learning approach. Transfer learning[5] is a machine learning technique that allows us to take advantage of models trained on generic tasks (in our case object recognition) and adapt them to a more specific task (in our case dog breed classification).

In this project, we develop and train a model for dog breed classification using transfer learning. We then use this trained model in a simple web app to predict the breed of an uploaded image of a dog.

Dog detection and human face detection can both be achieved using available pre-trained models.

Metrics

We use accuracy on the test datasets as our evaluation metric. Accuracy is often used for evaluating classification models[6]. It has the advantage of being very intuitive. It is computed according to the following formula:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

[2] https://github.com/udacity/deep-learning-v2-pytorch/tree/master/project-dog-classification

[3] http://vis-www.cs.umass.edu/lfw/

[4] https://flask.palletsprojects.com/en/1.1.x/

[5] https://en.wikipedia.org/wiki/Transfer_learning

[6] https://developers.google.com/machine-learning/crash-course/classification/accuracy

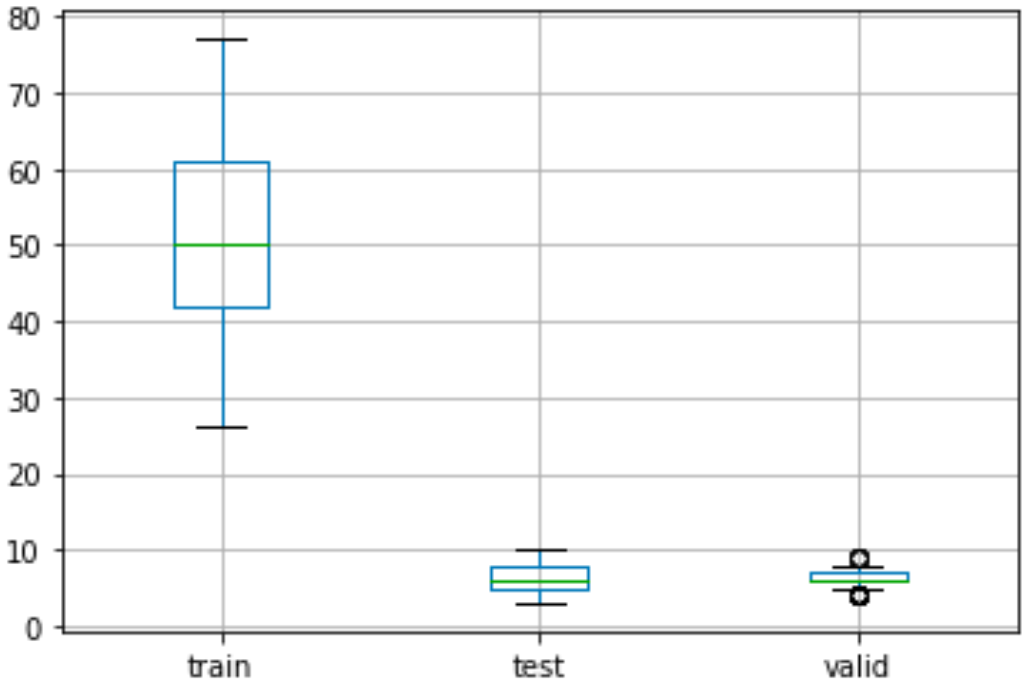
Analysis

Dog dataset: data exploration and visualization

The dataset contains 8351 images of dog belonging to 133 different dog breeds. Images are already split into train (6680 images), test (836 images) and validation (835 images) sets.

Let's look at the repartition of images in train, test and validation datasets:

	Train dataset	Test dataset	Validation Dataset
Mean number of images	50,22	6,29	6,28
Std	11,86	1,71	1,35
Min number of images	26	3	4
Max number of images	77	10	9



Number of images in the train, test and validation sets

We can see that classes are slightly imbalanced. However we have at least 26 images for each breed in the train dataset, which should be enough to start with.

Let's look at a sample of training images for the Belgian sheepdog class:

We can see that dogs are represented alone or in groups, in different positions and luminosity conditions and against different backgrounds. It seems that dogs 'faces are always visible. This might be an important feature used in our model. We can see that pups are represented in the dataset. So our model should be able to classify pups and not just adult dogs.

Algorithms and Techniques

As suggested in the notebook, we'll use different CNNs to classify dog breeds and detect dogs and humans:

Dog detector

We'll use a pre-trained VGG16[7] model to detect dogs. VGG models were first published in 2015 by Karen Simonyan and Andrew Zisserman. VGG16 was trained on the ImageNet dataset[8], one of the largest dataset of images. ImageNet contains ~14 million hand-annotated images. Annotations fall into ~20,000 categories such as «strawberry », « bow-tie » or in the case that interests us « dog ». A category typically includes several hundreds of images[9]. VGG16 achieved an accuracy of 92.7% when predicting the top-5 classes present in ImagesNet's images.

VGG16 is available in pytorch's torchvision models[10].

[7] https://arxiv.org/abs/1409.1556

[8] http://www.image-net.org/

[9] https://en.wikipedia.org/wiki/ImageNet

[10] https://pytorch.org/vision/0.8/models.html

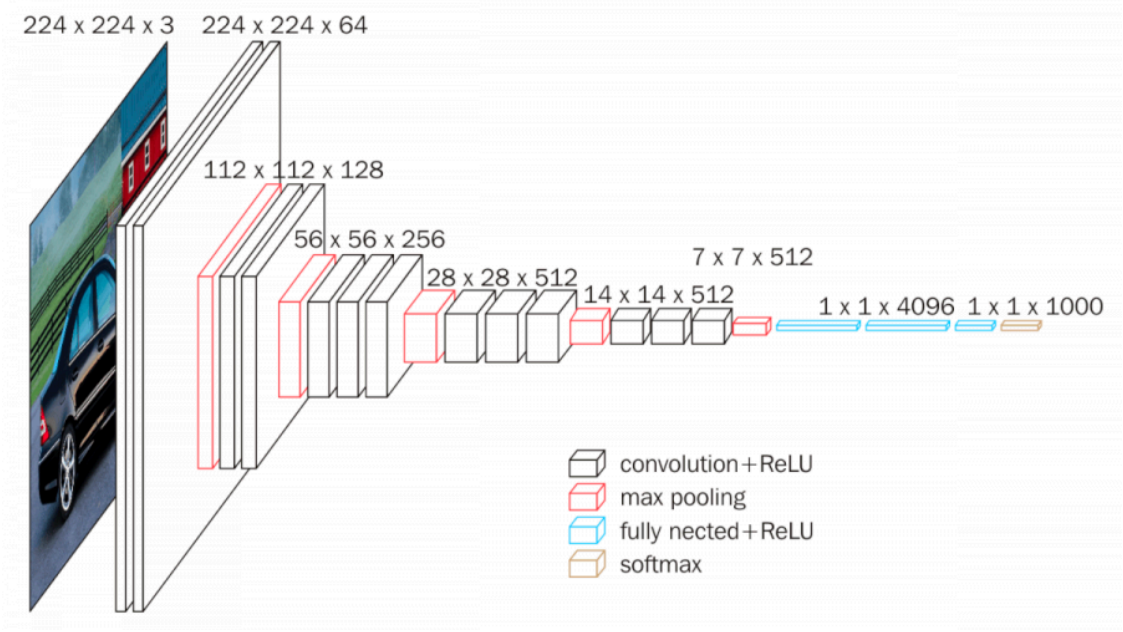


Image from https://neurohive.io/en/popular-networks/vgg16/

Human detector

We'll use OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images[11]. OpenCV provides many pre-trained face detectors, stored as XML files on github. We'll use the pre-trained face detector already available in the Udacity's dog classification project files.

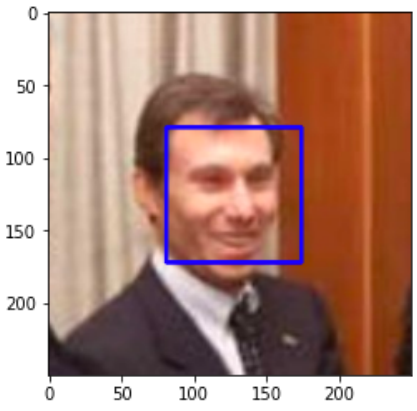


Image from https://github.com/udacity/deep-learning-v2-pytorch/blob/master/project-dog-classification/dog_app.ipynb

[11] https://docs.opencv.org/master/db/d28/tutorial_cascade_classifier.html

Dog breed classifiers

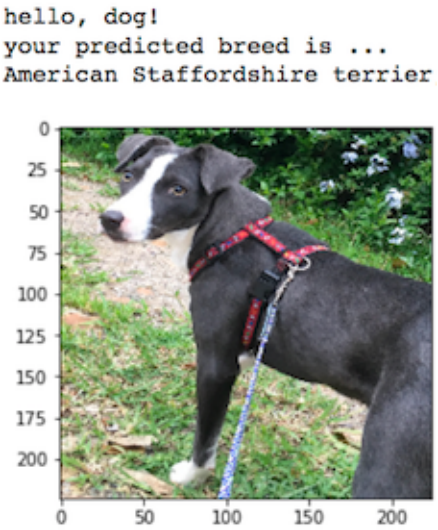


Image from https://github.com/udacity/deep-learning-v2-pytorch/tree/master/project-dog-classification

As suggested in the dog classification project notebook, 2 different models will be developed and tested to classify dog breeds:

- A CNN built from scratch, using a simple architecture derived from VGG16. This model will be developed inly for educational purposes. Such model is not expected to achieve a high accuracy for the task at hand.

- A transfer learning model using pytorch's pre-trained VGG16. As a first transfer learning model, we'll use a VGG16 architecture in which the last fully connected layer has been replaced with a fully connected layer where the number of output neurones = the number of dog breed classes present in the dataset (133).

Benchmark

We don't expect to reach a high accuracy for the CNN built from scratch. Classifying dog breeds is a challenging task because there can be considerable variability within a dog breed, and because some breeds are very hard to distinguish, even for humans. According to the project notebook « a random guess will provide a correct answer roughly 1 in 133 times, which corresponds to an accuracy of less than 1%. ». We'll follow the notebook, suggestion and be satisfied with an accuracy > 10%.

We found quite a few examples of transfer learning models for dog breed classification. Accuracy is often in the 40%-80% range[12][13][14]. We'll set ourselves an objective of accuracy>60%, as suggested in the notebook.

Methodology

Data Preprocessing

All images were normalized using Pytorch's recommended normalization: « All pre-trained models expect input images normalized in the same way, i.e. mini-batches of 3-channel RGB images of shape (3 x H x W), where H and W are expected to be at least 224. The images have to be loaded in to a range of [0, 1] and then normalized
using mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225]. »[15]

Training images

For a first attempt, I decided to randomly crop 224x224 pixels and randomly flip horizontally images in the train dataset. I did not revisit this choice of data augmentation processes since the model's accuracy was above threshold for the CNN from scratch and for the transfer learning model.

Test, validation, and input images

I resized images 224x224 px by first resizing the images to 256x256 pixels then center cropping to the target size.

[12] https://levelup.gitconnected.com/dog-breed-classification-using-cnn-and-transfer-learning-cc93a4497e90

[13] https://medium.com/@wangshuocugb2005/dog-breeds-classification-with-cnn-transfer-learning-92217cba3129

[14] https://towardsdatascience.com/dog-breed-classification-using-cnns-and-transfer-learning-e36259b29925

[15] https://pytorch.org/docs/stable/torchvision/models.html

Implementation and Refinement

CNN from scratch

I chose a simple CNN architecture with only 3 convolution units (convolution + Relu activation + max pooling) and 2 fully connected layers. I built a simple network based on the architecture of VGG16[16]. A quick google search of CNN used to classify dog breeds showed me that similar simple architectures could achieve an accuracy > 10%.

1st model:

- 3 convolutional layers with output channels = 64, 128, 256 respectively, padding=1 and stride=1.
- Maxpooling with pool size=2 and stride=2 after each convolutional unit.
- 2 fully connected layers, preceded by dropout layer with default value (0.5). The output number of neurones for the first layer is set to 4096.

Result: Out of memory error...

2nd model: Let's decrease dimensionality by using a stride=2 for the 3 convolutional layers and by using 2048 output neurones instead of 4096 in the first fully connected layer.

Result: Test Accuracy: 10% (86/836).

3rd model: Let's see if we can do better by increasing dimensionality. Let's set stride=1 for the 3rd convolutional layer. Let's also decrease the probability of dropout, p=0.2.

Result: Test Accuracy: 15% (129/836). Good enough!

Transfer learning model

I started with the smallest possible modification of the architecture of VGG16: I replaced the last fully connected layer with another fully connected layer with 133 output neurones (the number of dog labels) instead of 1000.

[16] https://www.jeremyjordan.me/convnet-architectures/#vgg16.

Results

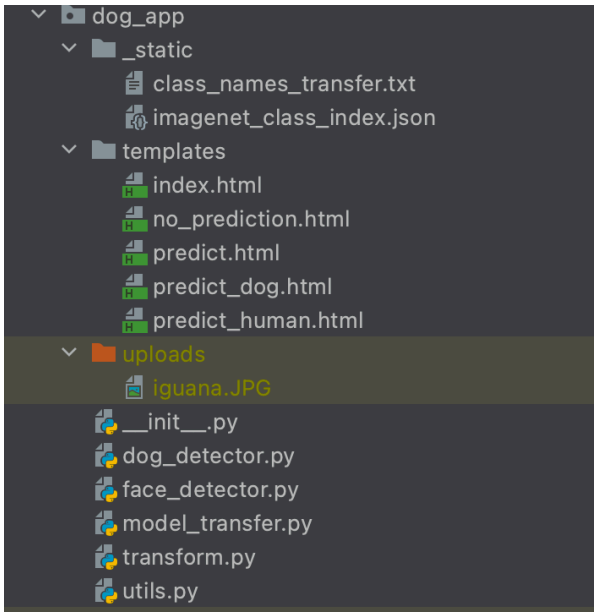
We'll use the transfer learning model in our web application. This model achieved an accuracy above threshold (accuracy=80%, i.e. 671 correct predictions/836 images in the test dataset). This is a much higher score than the CNN from scratch (accuracy=15%, i.e. 129 correct predictions/836images in the test dataset). And much higher than the accuracy of a random prediction (accuracy<1%, i.e. 1/133 classes).

For our web application project, an accuracy of 80% is satisfying. The penalty associated with making an incorrect prediction is after all low.

Deployment

We integrate the dog detector, human face detector and dog classifier using transfer learning into a simple web application using Flask.

Here is the structure of the application:



- __init__.py is the entry point for the application. It contains configuration and routes.
- dog_detector.py contains the code for the dog detector, taken from the notebook
- face_detector.py contains the code for the human face detector, taken from the notebook
- model_transfer.py contains the code for the dog classifier. It uses the transfer learning model's state dictionary, obtained by executing the notebook.

- transform.py contains the code for preprocessing uploaded images, taken from the notebook
- utils.py contains utility functions such as a function to check uploaded files extensions or a function to clean up the uploads/ folder.
- The _static/ folder contains class names for the transfer learning models and for the dog detector.
- The templates/ folder contains html templates used to render the application.
- The uploads/ folder is used to store the last uploaded image

Instructions for running the application can be found in the README.md file.

Here are some examples of predictions:

File Upload

Choisir un fichier Aucun fichier choisi

Submit

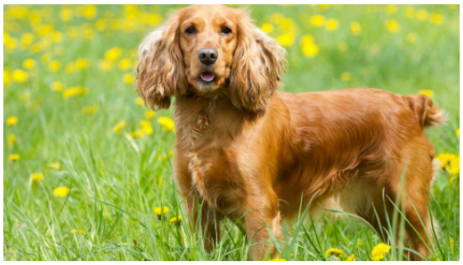
Result

Hello dog! I think you are a ... Australian shepherd

File Upload

Choisir un fichier Aucun fichier choisi

Submit



Result

Hello dog! I think you are a ... English cocker spaniel

File Upload

Choisir un fichier Aucun fichier choisi

Submit

Result

Hello dog! I think you are a ... Nova scotia duck tolling retriever

File Upload

Choisir un fichier Aucun fichier choisi

Submit

Result

No dog or human detected.

File Upload

Choisir un fichier Aucun fichier choisi

Submit

Result

Hello human ! Your predicted dog breed is ... Pharaoh hound

File Upload

Choisir un fichier Aucun fichier choisi

Submit

Result

Hello human ! Your predicted dog breed is ... Irish water spaniel

Note that the case of multiple dogs of different breeds is badly handled. This could be an interesting subject for future work.

Future work

Possible avenues for improvements:

1. Improve prediction accuracy by trying different data augmentation techniques, model architectures or parameter values.
2. Handle the case of a picture with dogs of different breeds. In the current version only one breed is predicted.
3. Display a confidence score for the prediction. If the confidence score is low, display the top 3 guesses. This could be relevant in the case of mixed dog breeds.