11 May 2020
Marianna Janczak

Capstone project

Machine Learning Engineer Nanodegree Project

CNN project: Dog Breed Classifier

I. Definition

1. Project overview

Mobile apps making use of machine learning algorithms are becoming more and more popular. They often utilize speech recognition and computer vision to offer richer and more user friendly experience. They can help people with disabilities as well as provide entertainment.

Image classification tasks are an important part of machine learning. They range from medical scans analysis to self driving cars. Numerous approaches have been developed, to tackle this problem, from simpler ones like basic feature extraction (e.g. overall brightness or gue of the picture) to deep convolutional neural networks.

As I have always been interested in the fun part of using machine learning, I decided to take on a project, that can join education with entertainment and prepare a model, that is able to recognize dog breeds from pictures. It can also detect humans on the provided pictures and suggest the most similar dog breed.

This is a Udacity specific project, which means that is has an already prepared data set and a set of instructions, guiding through different possible implementations and exploring the subject of convolutional neural networks. The main repository for the project can be found on GitHub[1].

2. Problem statement

This project aims at constructing and training a machine learning mechanism, that is able to classify images. Any appropriately preprocessed image, given to the mechanism should result in one of three possible outputs:

a) if the image is a picture of a dog, the mechanism should output information about the breed of the dog

b) if the image is a picture of a human, the mechanism should suggest what dog breed is most resembling the shown human

c) in any other case the mechanism should inform user, that the image contains neither a picture of a dog nor of a human.

This is in fact a multi class classification task, aimed at correctly classifying images according to a dog breed of a dog pictured on them.

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

3. Metrics

To quantify the performance of the models, accuracy was used. As the project in reality consists of two separate models (one for human detection and one for dog breed classifier), both models are evaluated separately.

Accuracy is the most basic metric that can be easily applied to classification problems. It takes into account both true positives and true negatives.

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

II. Analysis

1. Data Exploration

As this is a Udacity specific project, the dataset for it has already been provided. There is no indication, as to the source of the images. The dataset consists of two parts:
a) dog dataset - 8351 images divided into 133 folders, each corresponding to a different dog breed. Those are color images in jpg format, with different resolutions, apparently not normalized in any way.

Fig. 1 *Example images from dog data set*

b) human dataset - 13233 images of humans divided into 5749 folders, each containing one person's photos. The folders are named according to the names of persons on the pictures. The pictures are color images in jpg format normalized to the resolution of 250 x 250, with black padding where it was necessary.

Fig. 2 *Example images from human data set*

Both human and dog images have to go through a suitable preprocessing. After that, the human images are used for the human face detector with the use of OpenCV library and the dog images are the input for classification models, based on CNNs and transfer learning.

2. Exploratory Visualization

The dog dataset is divided in three parts - the main one of 6680 images for training, 835 for validation and 836 reserved for testing purposes.
The training data is rather balanced among the 133 classes, with the average of 50 images per class, the maximum being 77 and minimum 26.

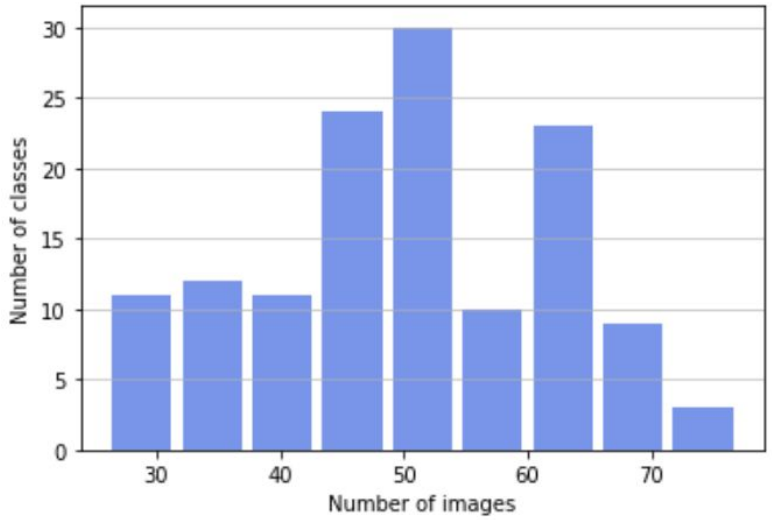


Fig. 3 *Dog data set distribution*

The human dataset can be also deemed balanced with only 143 outliers (established as more than 10 photos of one person). Most of the folders contain less then 10 images of one person and the average without the outliers is 1.62. After removing the outliers, most of the folders contain 1 or 2 images. As the dataset is mostly used for showing the functioning of the human detector, all the images are left in the dataset.

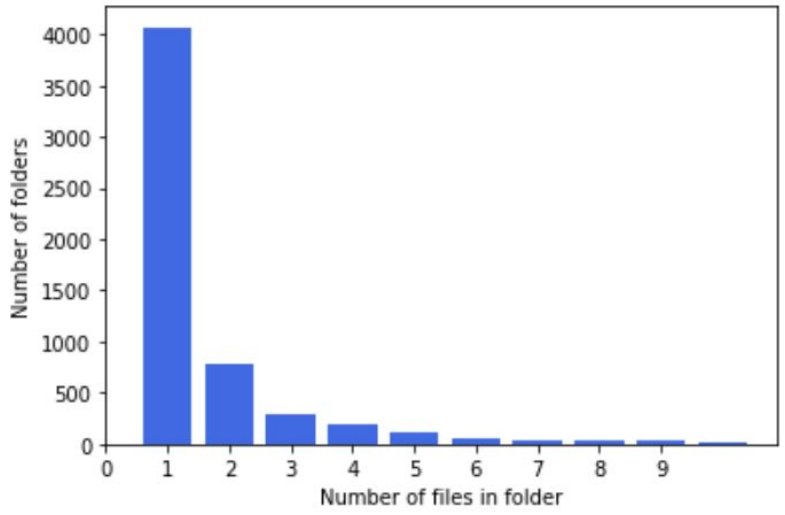


Fig. 4 *Human data set distribution*

3. Algorithms and Techniques

As the project has two distinct parts, there are two main algorithms to be used in it.

For the human detector a pretrained Haar feature-based cascade classifier is used. This is an efficient algorithm for object detection in images. It was first described in the paper *'Rapid Object Detection using a Boosted Cascade of Simple Features'* [2] by Paul Viola and Michael Jones. The main concept behind cascade classifiers is performing object detection in several distinct steps. In each step a set of features is used to estimate, if in certain region of the picture the important element is present. If not the image is discarded.

The main dog breed classifier model is created with the use of transfer learning. "Transfer learning is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned."[3] This is motivated by the fact, that with a relatively small dataset training a model from scratch to recognize 133 distinct classes would not be effective. There have been attempts at solving this kind of task with CNNs, detecting facial keypoints and feature extraction, as presented in *"Dog Breed Identification"* paper by Whitney LaRow, Brian Mittl and Vijay Singh. But their best classifier, as they write, *"predicts the correct dog breed on its first guess 52% of the time; 90% of the time the correct dog breed is in the top 10 predictions"*.

The main feature extraction part is based on a pretrained model, that was trained on a large dataset. Only the last layers (the classifier) of the original model are exchanged for the layers trained with the dog breed dataset. This is based on the ability of deep convolutional layers to learn to extract features from images. So what we in fact accomplish by using transfer learning is using powerful feature extractor from a pretrained model and only training a linear, fully connected layers based classifier, to work on these features, to perform the classification according to the new requirements.

4. Benchmark

As this is transfer learning based approach to a classification problem two different benchmarks were used, to show how the different parts of the model work.

For the feature extractor part as a benchmark is used a pretrained VGG-16[4] model, that is available as a part of PyTorch library[5]. The model is pretrained on ImageNet[6]. This is a large image dataset, separated into 1000 classes, indexed 0-999. Images with different dog breeds are in classes with indexes 151 and 268 (inclusive), which allows to use it as a basis for a dog detecting model.

For the classifier part a simple CNN model built and trained from scratch is chosen as a benchmark. This is to confirm, that witch such a small dataset transfer learning is indeed required.

[2] https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf
[3] Chapter 11: Transfer Learning, Handbook of Research on Machine Learning Applications, 2009
[4] https://arxiv.org/abs/1409.1556
[5] https://pytorch.org/docs/master/torchvision/models.html
[6] model. http://www.image-net.org/

III. Methodology

1. Data Preprocessing

As there are two separate models in the whole mechanism, there are in fact two image preprocessing pipelines, that allow the dog classifier and the human detector part to run smoothly.

For the human detector the pipeline was determined by OpenCV HaarCascade requirements for image processing. This involves reading in the image with imread CV2 function and converting it to grayscale.

For the classifier the preprocessing pipeline is much more involved, as it needs to normalise the images in a way,that they can be converted into Tensors, that are used by pyTorch models. This pipeline is different for training and prediction phase, as for training it involves data augmentation techniques, which are not needed in the prediction pipeline. All preprocessing steps are done using PyTorch's torchvision.transforms[7]

a) training pipeline preprocessing:

Being random these methods allow for data augmentation, as they can create different images from one image.

1. Random Horizontal Flip - randomly flip the image horizontally
2. Random Rotation - rotate the image by a certain degree, in range from -20 to 20 degrees.
3. Random Resized Crop - first create a random crop from the input image and then resize it to a specified value - in this case 224 pixels x 224 pixels.

b) prediction preprocessing pipeline:

The prediction preprocessing does not involve any random methods. Instead the preprocessing aims at best preserving all the information in the image. That is why the whole image is first resized, and then cropped from the center, as usually, there is more information in the center, than on the edges of an image.

1. Resize - resize the image to 255 pixels x 255 pixels
2. Center Crop - crop the image from the center to 224 pixels x 224 pixels

Both pipelines end with the same step - turning the image into a Tensor, to be suitable for PyTorch models.

2. Implementation

The implementation process follows the design of the project and is carried out in the following steps:

a) Creating a simple human detector.

This was done with the use of pretrained HaarCascade available from OpenCv library: haarcascade_frontalface_alt[8]. As this is a Udacity specific project, the part of the code for reading in the model was provided. After reading in the model the face_detector function was created, that for a given path to an image file performs first the necessary data preprocessing and then uses the HaarCascade to find faces in the given image. The function returns true if any human face was found and false otherwise.

b) Creating a dog-detector.

A pretrained VGG-16 model is used just to detect presence of a dog in a given image. This is a Udacity supplied code, that demonstrates the robustness of pretrained models.

[7] https://pytorch.org/docs/stable/torchvision/transforms.html

[8] https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_alt.xml

c) Creating a benchmark classifier

A reference classifier is built from scratch. This is a simple CNN model. There are only three convolutional layers, the first one starting with 3 input channels (as the images in the dataset are RGB, so they have 3 channels), the output being 16 channels, and the next 2 layers doubling the number of channels. The padding of 1 insured, that no data from the images would be lost. The pooling layers allowed for reducing the dimensions of the images, cutting down the number of parameters, which helped to make the learning process faster.

There are only 2 fully connected linear layers, with the dropout of 0.25 implemented before each layer. This helped to avoid some of the neurons getting too much influence on the whole model.

All the layers (except for the last one) use ReLu activation function to help to normalize the weights.

d) Creating and training the main dog breed classifier.

As could be easily seen from the previous benchmark model with a classifier built from scratch, with this small amount of data a different approach to building a robust model had to be used. Transfer learning is a good choice in this kind of tasks, as it helps to leverage the power of models trained on huge datasets, at the same time giving the possibility of many customizations.

One of the most important decisions is the choice of the pretrained model, that will serve as a feature extractor for the custom classifier. In this project a pretrained Resnet-152[9] is used as the base model. Resnet models were designed to cope with the problems involved with really deep neural networks, like the vanishing gradient problems. Therefore they are well suited to cope with many convolutional layers needed to extract features from images. The pretrained Resnet-152 made use of the ImageNet dataset, which ensures that the convolutional layers were properly trained to extract features from images.

The classifier part of the model is relatively simple. It consists of one linear, fully connected layer with ReLu activation function and 2048 inputs, followed by another linear, fully connected layer with LogSoftmax activation function and 133 outputs. The number of inputs is dictated by the Resnet-152 architecture, and 133 outputs relate to 133 classes, that the model has to predict. The LogSoftmax activation function ensures, that the resulting outputs are log-probabilities of the pictured dog belonging to each class (dog breed).

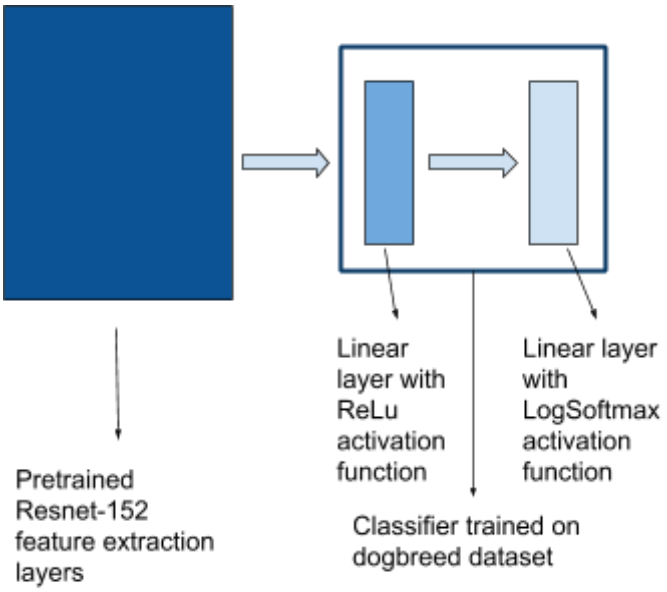


Fig. 4 *Main dog classifier model architecture*

[9] Resnet-152 model https://arxiv.org/abs/1512.03385

For the training of the model the following parameters were used:

1. As a loss function negative log likelihood loss was chosen. It is particularly well suited for multi class classification problems, as it operates on log-probabilities for each class.
2. For the optimization algorithm the PyTorch implementation[10] of Adam optimizer was chosen. This was dictated by the robustness and ease of use of this algorithm, as described in 'Adam: A Method for Stochastic Optimization'[11] paper by Diederik P. Kingma and Jimmy Ba.
3. Learning rate: 0.001
4. Number of epochs - 10
5. Batch size: 20

To perform the training of the model the dataset was split into three parts: training, validation and test. The test part was set up for the final model evaluation, while the validation dataset was used during training to check the model loss decreasing during training. Each part was preprocessed with appropriate preprocessing pipeline and then turned into a PyTorch DataLoader[12], which combines a dataset with a sampler. This provided the means of feeding the data into the model in batches, in an organised and efficient manner. It also provides shuffling of data, which provides the means of combating overfitting.

The training was done on a GPU for 10 epochs.

e) Creating final output mechanism

The goal of the project was constructing a mechanism to provide answers on estimated dog breed of a dog in the picture, and if a human was present, to give the most similar dog breed name. To accomplish this a function was created, that takes a path to an image as an input. It then calls the dog detector and if there is a dog, it calls the dog classifier prediction method and outputs the estimated dog breed. If there is no dog on the picture, the human detector is run, to check if there is a face on the picture. If there is, a the image is run through the dog classifier again, and the most similar dog breed is output. If there is no human on the picture, appropriate info is returned.

3. Refinement

Many factors being determined by the choice of the base model and the character of the task. Number of inputs and outputs of the classifier was determined by the architecture of the chosen base model and number of classes. The NLL function was optimal for multi class classification as it operates on log probabilities.

First experiment with learning rate at 0.00001 yielded no results, as the training was taking too long on the available GPU. So the learning rate was set at 0.001 which is a common choice in this kind of machine learning tasks. The batch size of 20 was a reasonable compromise between the models capabilities of generalisation (the bigger batch size the better) and the time efficiency of training and hardware limits.

Ten epochs were also chosen as a reasonable value. As the checkpoint for the model parameters was saved every time value loss was smaller than previously, the resulting trained model was the best it could get in the chosen number of epochs. If it hadn't achieved the necessary accuracy, it would be possible to train it for more epochs.

[10] https://pytorch.org/docs/master/optim.html#torch.optim.Adam
[11] https://arxiv.org/abs/1412.6980
[12] https://pytorch.org/docs/master/data.html#torch.utils.data.DataLoader

IV. Results

1. Model Evaluation and Validation

The main dog breed classifier was evaluated in three ways:

1. During training, the validation dataset was used after each epoch, to calculate the validation loss. This clearly showed, that the model is learning properly and not overfitting, as the validation loss was getting smaller, in parallel with the training loss. Additionally during training a mechanism was set up, so that a checkpoint with model weights would be created every time the validation loss got smaller, so the best model weights were chosen for the prediction function.
2. The test dataset was used to count the accuracy of the model. The accuracy of 86% is not a bad result if we consider how small the data set is.
3. Several human and dog images were processed and the results were visualised. It is clearly visible, that the breed of the dogs on the pictures is right. As to the most similar dog breed for the human faces it may be subjective, but the results were satisfactory and made sense.

2. Justification

The dog breed classification model was compared to two different benchmark models.

a) A pretrained VGG-16 model was used as a dog detector. It reached 99% accuracy, but it's task was much easier, as it in fact worked as a simple binary classifier (something is a dog or not). This however showed how robust a pretrained model can be with extracting the necessary features for the more complicated multiclass classifier. This is the main reason, that with a small dataset, the proposed dog classifier could reach 86% accuracy in predicting 133 classes.

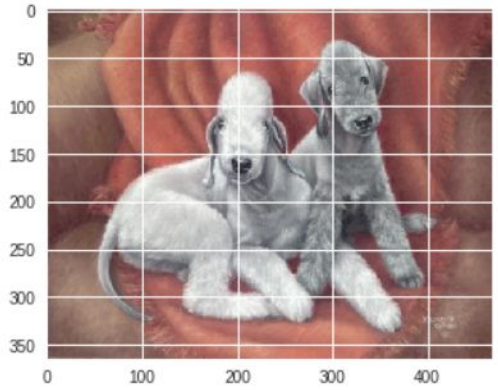
b) A convolutional neural network built from scratch (*for the architecture see Chapter 'Implementation', c) Creating a benchmark classifier*). This CNN model was trained only on the provided data set for 100 epochs. The size of the model and the number of epochs were a reasonable setup for the time and computation capabilities for a benchmark model. This model attained only 14% accuracy, which means it's roughly 6 times less accurate than the final dog breed classifier model.

V. Conclusion

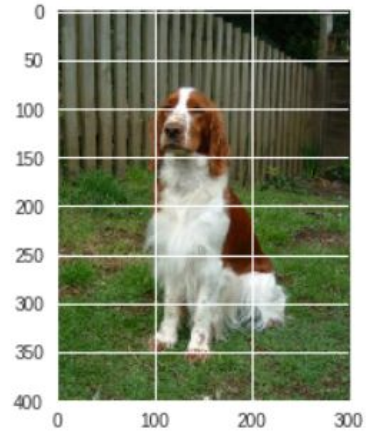
1. Free-Form Visualization

The mechanism was run on several dog and human images.

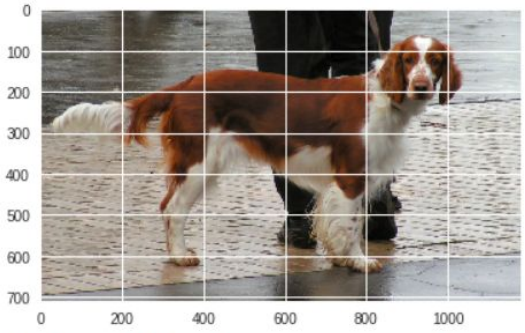
It is clearly visible that the do pictures are correctly labeled with the dog breed:



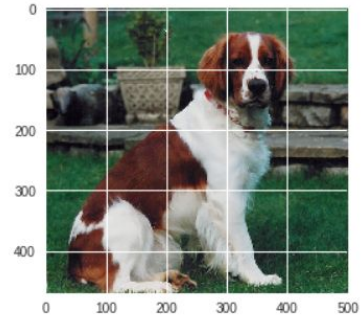
This looks like Bedlington terrier



This looks like Welsh springer spaniel

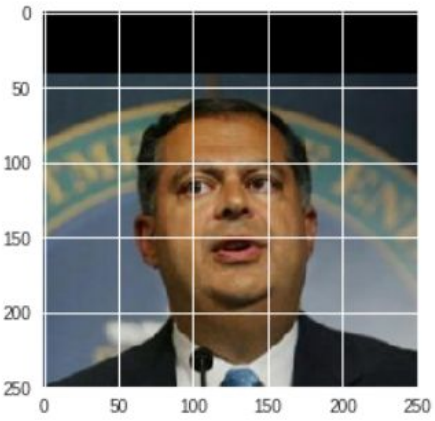


This looks like Welsh springer spaniel

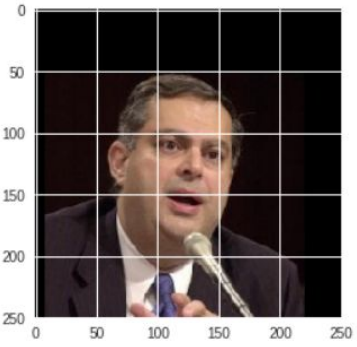


This looks like Welsh springer spaniel

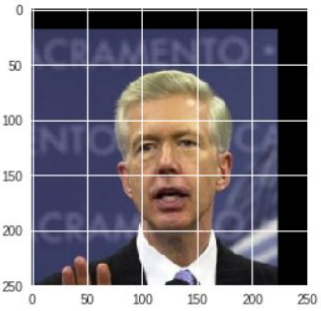
The dog breed similarity of human faces if subjective, but the model output good estimates. For reference a picture of a dog of the estimated breed, taken from the test dat set is shown next to the model output.



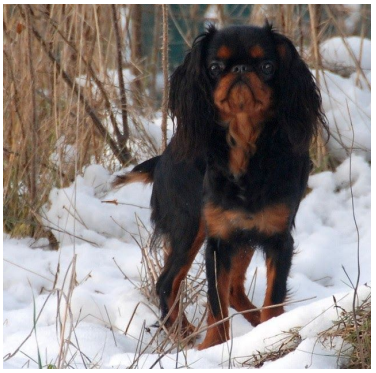
This human remarkably resembles Bearded collie



This human remarkably resembles Chinese crested



This human remarkably resembles English toy spaniel



2. Reflection

As a whole this was a really interesting and involving project. Although it is a Udacity specific project, with provided notebook guiding through the relevant steps, it proved a great way to explore the subject of convolutional neural networks and transfer learning.

The whole process for this project follows a clear and logical structure, divided in the following steps:

a) Data exploration and preparation.

This is a very important step in any machine learning model, as usually the type of solution greatly depends of available data. In this case the data was already provided, but it had to be properly prepared to be suitable for use wuth machine learning models.

b) Preparing human detector.

In this step there was a chance of familiarising with OpenCv library, which is one of the most popular tools in any computer vision related tasks and the use of pretrained Haar Cascade classifier was a great primer for object identification algorithms.

c) Constructing a simple dog detector on the basis of a pretrained model.

Here there was a chance for familiarising with some really important concepts, that were invaluable in later parts of the projects. The pretrained models that are available from PyTorch are an excellent basis for transfer learning. Also the ImageNet dataset was introduced, which can be useful in any future computer vision tasks.

d) Constructing a basic CNN classifier

This step was crucial to understanding the way convolutional neural networks work. Although the model is rather simple, it nonetheless shows how the convolutional layers work, and how they are linked with fully connected, linear layers, to form a classifier.

e) Constructing the main dog breed classifier.

This is the most advanced part of the project. It combines the knowledge gained in previous steps with the transfer learning concept. Thanks to the robustness of a model pretrained on a huge data set, a simple classifier added to it is enough to reach over 80% accuracy on multi class classification, while using a relatively small dataset.

f) Combining all the concepts

For the project to conclude a simple function was made, that was able to use the capabilities of the human detector, dog detector and dog classifier to give the desired output, forming a basis for an app, that could be possibly deployed and use in a mobile application.

3. Improvement

Although the model performs relatively well, concerning the small dataset it uses, there are several improvements, that could be applied.

a) A better accuracy could be reached for the dog classifier. This could be achieved in several ways, using all or only some of them, according to tim e and hardware capabilities:

- training for more epochs
- using a bigger data set
- using a bigger batch size

b) Getting better and more efficient human detector and dog detector, which could make the app run faster and consume less resources.

c) Possibility of deploying the whole mechanism as a Flask app and using it in a mobile application.

VI. References

1. Udacity Dog Breed Project Github Repo
https://github.com/udacity/deep-learning-v2-pytorch/tree/master/project-dog-classification
2. Paul Viola and Michael Jones, 'Rapid Object Detection using a Boosted Cascade of Simple Features'
3. Chapter 11: Transfer Learning, Handbook of Research on Machine Learning Applications, 2009
4. VGG-16 https://arxiv.org/abs/1409.1556
5. PyTorch pretrained models https://pytorch.org/docs/master/torchvision/models.html
6. ImageNet dataset model. http://www.image-net.org/
7. PyTorch torchvision.transforms https://pytorch.org/docs/stable/torchvision/transforms.html
8. OpenCV haarcascade_frontalface_alt
https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_alt.xml
9. Resnet-152 model https://arxiv.org/abs/1512.03385
10. PyTorch Adam Optimiser https://pytorch.org/docs/master/optim.html#torch.optim.Adam
11. Diederik P. Kingma and Jimmy Ba, 'Adam: A Method for Stochastic Optimization'
https://arxiv.org/abs/1412.6980
12. PyTorch DataLoader https://pytorch.org/docs/master/data.html#torch.utils.data.DataLoader