Machine Learning Engineer Nanodegree, Udacity

Capstone Project

Christophe BOURGOIN
February 11th, 2020.

Building a Dog Breed Classifier using Deep Learning

I. Project Definition: Project Overview, Problem Statement & Metrics

A. Project Overview

Even for a passionate & specialized human person on dogs as a veterinary, it's often difficult to identify perfectly the dog breed. A mobile or web app that would be able to determine the dob breed from an image would be therefore particularly useful for a lot of people.
With the incredible development of deep learning & computer vision in the last years, it's now possible to create this kind of app. This is what I propose to do in my capstone project.

More precisely the purpose of this project is to build algorithms that are able to detect & then classify automatically the dog breed from a picture. Such algorithms could then deployed easily in an app.

Using 2 datasets which are provided by Udacity & directly accessible in the Udacity's workspace (a first dataset composed a 13233 human images & a second dataset composed of 8351 dog images), I developed a series of models that could be used as part of a mobile or web app & that to perform if a dog or a human is detected in an image and then to provide an estimate of the dog's breed if a dog is detected or an estimate of the dog breed that is most resembling if a human is detected. If neither a dog or a human is detected in the user-supplied image, an error message will be sent.

B. Problem Statement

As explained above, in this project I'll build a series of algorithms that detect the presence of a dog or a human in an image & then provide an estimate of the dog breed (or the dog breed that is the most resembling if a human is detected). A critical part of this problem is, therefore, to classify correctly the dog breed.
More precisely, to approach my problem of dog breed classification from an image, I'll build a machine learning pipeline which would be composed several steps :

1. **Data exploration** :

This step will aim to understand the main characteristics of used data, and especially to appreciate the quality of images in datasets.

2. **Data Preprocessing & Partitioning** :

Then I'll apply some transformations to the images in order to improve the quality of my models. This step is crucial to avoid the "Garbage In Garbage Out" principle. At this step, I'll split also the images in our dataset in 3 parts : a training part, a validation part & a test part. This will be very useful to train & validate my model in the step 4 of my ML pipeline and this will allow me to select the best model to predict dog breed.

3. **Definition of my model architecture**

As I think it will be judicious to use a deep learning model to predict dog breed from a images, especially a convolutional neural networks (I'll explain more precisely this kind of neural networks in the section Algorithms & Techniques), I'll define at this step the characteristics of its architecture : its layers, its size & depth, what kind of activation functions I'll use …

4. **Train & Validate my model**

This step I'll train my model in order to it learns from data & compute the optimal weights & parameters. A this step, I'll be able to validate a model architecture that should perform well.

5. **Test my model**

Once a model architecture has been validated, I could test my model on unknow images & to measure its performance from a success metrics (in our problem, its accuracy). A this end of this step, I'll know what is the performance of my model & if it performs well to predict dog breed.

6. **Using my model to predict dog breed from new image**

Finally I could use my model on new images & obtain a dog breed prediction for each of them.

To note that my ML pipeline will not a linear or sequential process but I should test differents versions of my pipeline & each time, to assess if my hypothesis & choices improve my results. So **my ML pipeline would be a long iterative process** with back & forth between my different stages, especially between my step of model architecture definition and my step of training & validation model.

C. Metrics

To measure the performance of my model, I computed the accuracy. In its crash course of machine learning, Google explained, for example, that "Accuracy is one metric for evaluating classification models. Informally, **accuracy** is the fraction of predictions our model got right. Formally, accuracy has the following definition":

Accuracy = Number of correct Predictions / Total number of predictions

In the context of our problem, I don't think it will be necessary to compute other performance metrics like recall. But in other contexts, like a medical context, it's often necessary to control false negative or false positive.

II. Analysis :

A. Data Exploration & Visualization

Before to build my models of detection & of classification, I analyzed our dataset of images. Below are 2 samples of images.

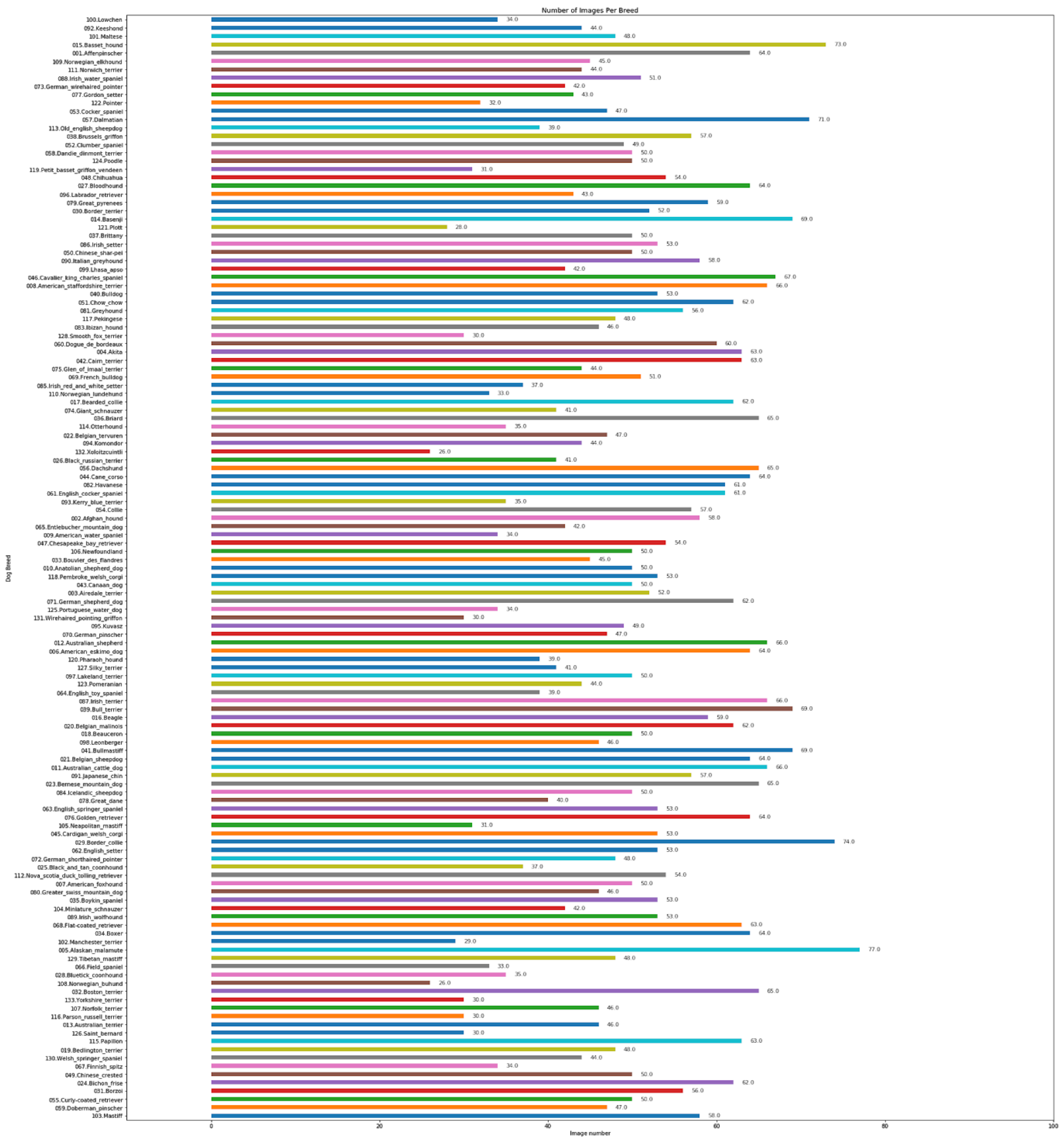
Fig 1: Sample of human images

Fig 2: Sample of dog images

Before to build a classifier, I paid especially a particular attention to your dataset of dog images. Notably, I paid attention to 2 dimensions: the number of dog images per breed & the size of dog images.

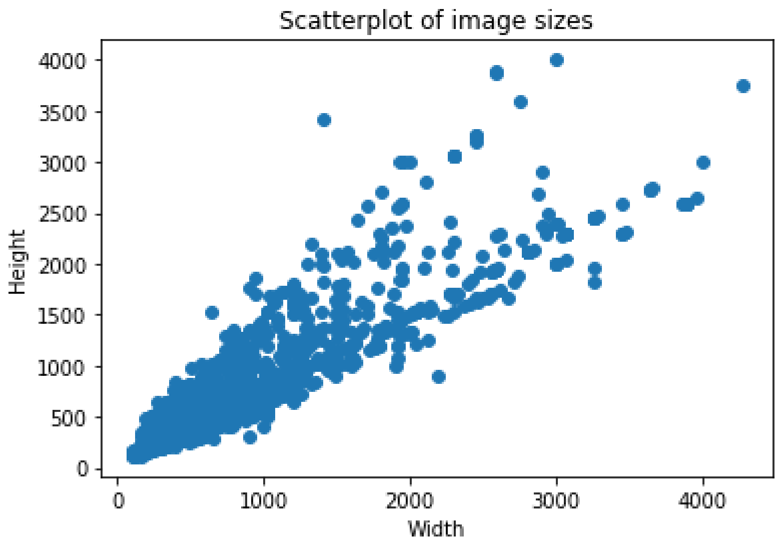
From the analysis of the distribution of dog images number (see below, for example, the figure), even if I noted that certain breeds are more represented in our dataset than others, I expect that this should not have an impact on my results. Ideally,
I'll analyze my results of model to check if I'm able to detect some curious predictions.

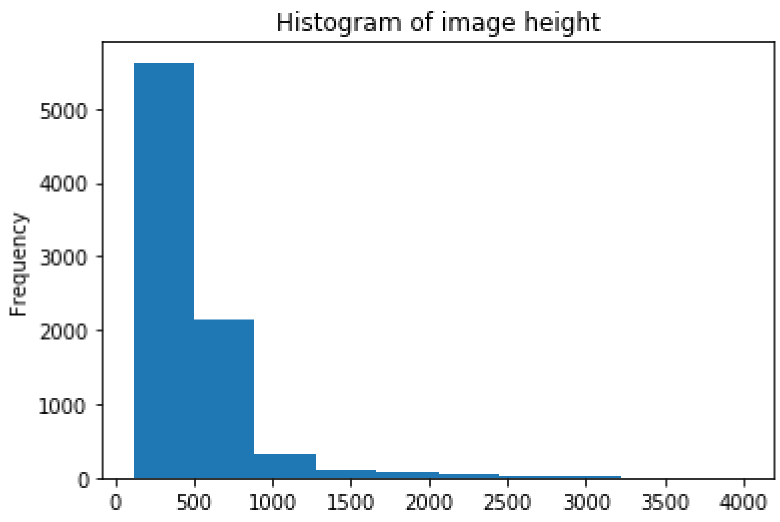
Fig 3 : Number of dog images per Breed

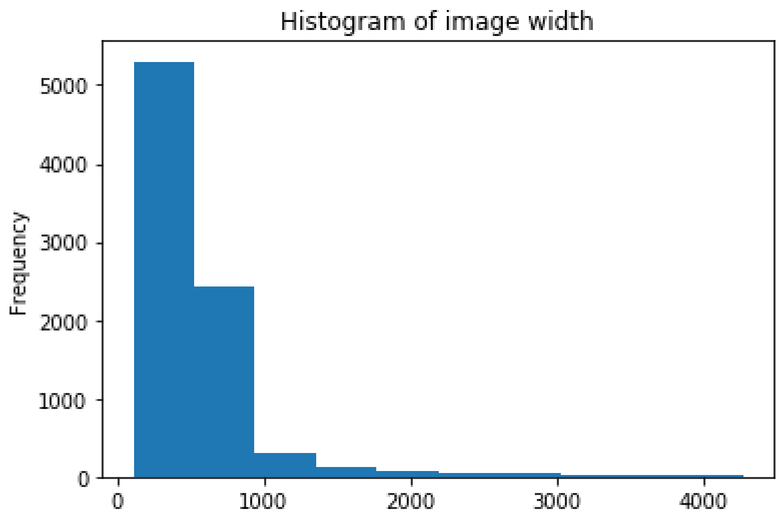


Then I got interested in the sizes of our dog images. In my exploratory analysis, I build a dictionary with the different shapes of our images, defined by OpenCV as the following tuple (rows, columns, color channels)) in keys & the occurrence of each size in values.
I concluded that our dataset is composed of a very large number of image sizes. More precisely, I computed precisely by 4216 different image shapes. To illustrate the heterogeneity of image shapes, below are a scatterplot of image sizes & 2 histograms of image height & width.

Fig 4 : Scatterplot & histograms of image sizes







Therefore, I expect it will be necessary & useful to resize each image before to use them in my models.

B. Algorithms & Techniques

My solution will be composed of 2 steps:

1) in a first step, I'll start to build algorithms to detect human or dog faces.

To build an algorithm of human face detection I'll use OpenCV & Dlib libraries. My second version of the model from the Dlib library should perform very well on our human dataset while reducing the false positive of detected dog face in the dog dataset.
Then to detect the dogs in images, I'll develop a dog detector from a pre-trained model. I used a model that is called VGG16. VGG16 is a convolutional neural network having been trained on ImageNet, which is a very large dataset used for image classification & other vision tasks. VGG16 model is able to classify an object in a picture in one of its 1000 categories.

Below is the VGG16 model architecture :

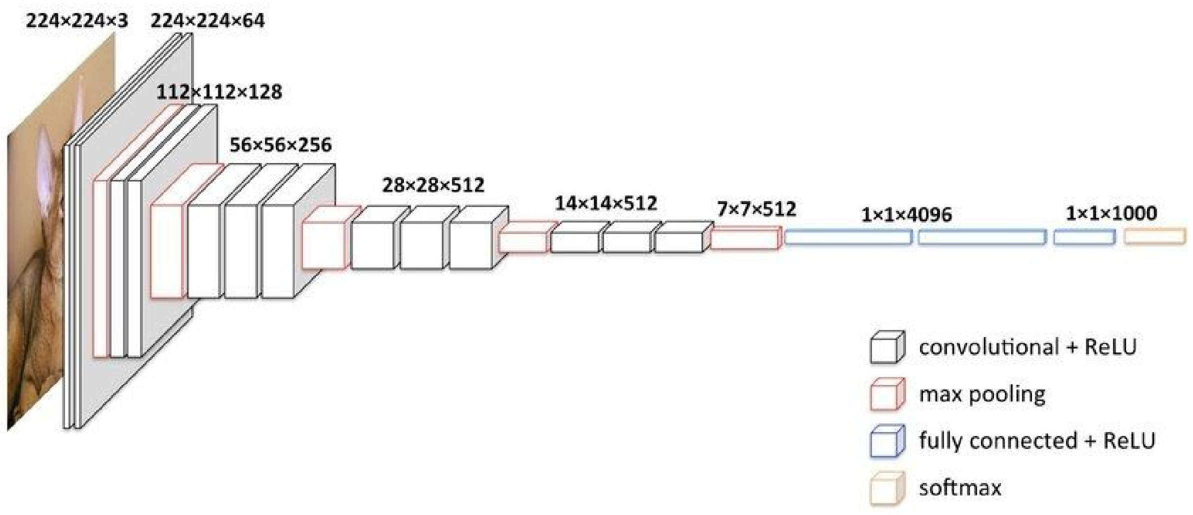


Fig 5: VGG16 model architecture[1]

In our project, I'll use the VGG16 model to identify & detect a dog in an image.
To sum up, the goal of this step will be able to detect if a dog or a human is present in an image.

2) In a second step, I'll develop an algorithm to predict the most probable dog breed from images.

More precisely I decided that I'll build convolutional neural networks (ConvNet or CNN) to predict & classify dog breed from images because the CNNs represent the state-of-art for this kind of image classification task. As described by Sumit Saha[2], a CNN is a deep learning algorithm which can take in an input image, assign importance (learnable weights & biases) to various aspects/objects in the image and be able to differentiate one from the other.
A ConvNet is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights.
An important part of a CNN is its convolution layer (conv2d layer in Pytorch) because this kind of layer is able to extract the high-level features such as edges from the input image.
Below is an illustration about the convolution operation[3] :

[1] "Learning better deep features for the prediction of occult invasive disease in ductal carcinoma in situ through transfer learning", Shi & al (2018), https://www.researchgate.net/publication/323440752_Learning_better_deep_features_for_the_prediction_of_occult_invasive_disease_in_ductal_carcinoma_in_situ_through_transfer_learning

[2] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[3] https://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/

Fig 6 : The Convolution Operation

In this second step, I'll start to build a **first convolutional neural network from scratch** based on the following architecture : 3 conv2d layers & 2 linear (fully connected) layers. I applied a relatively standard filter selection from 32 to 64 to 128 and I also introduced non linearity in adding ReLu operation after every convolution. Moreover, between each layer I applied max pooling to decrease the dimensionality of each feature map but to retain the most important information. To better control overfitting of my model, I added dropout that will drop out nodes randomly during training.

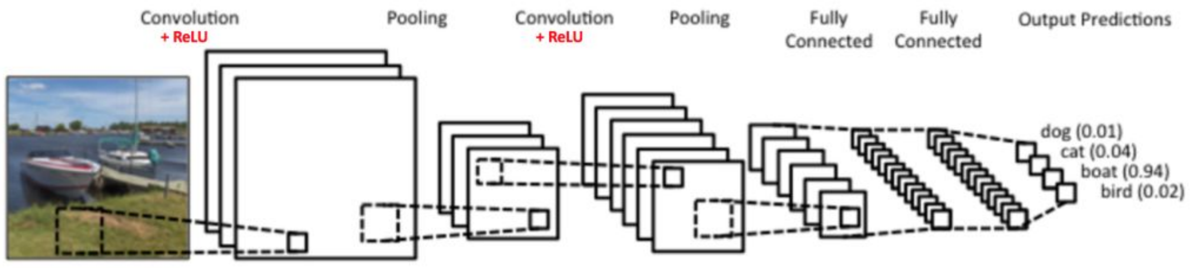


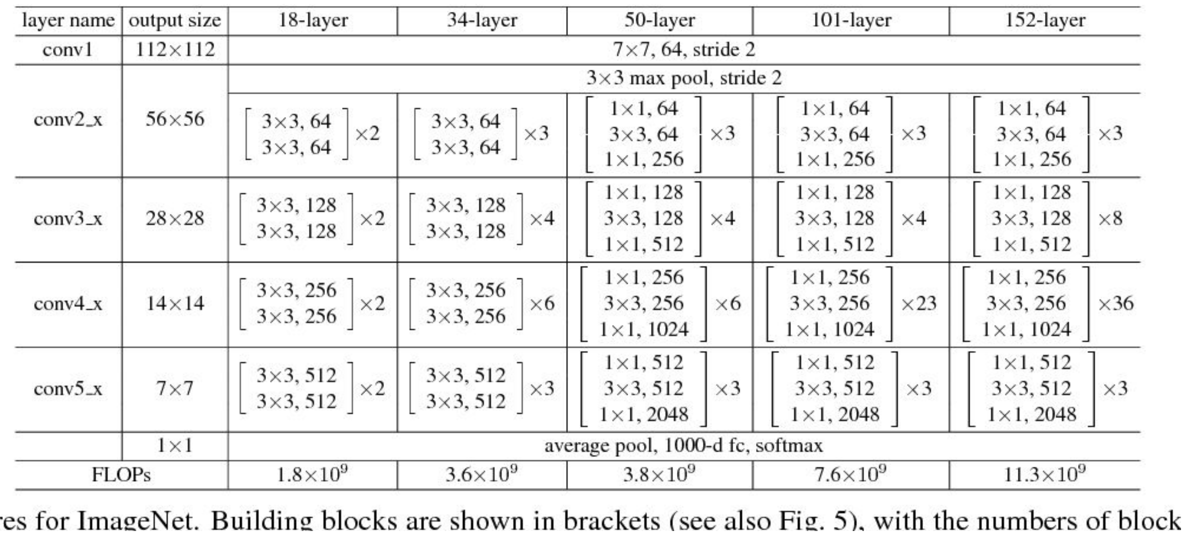
Fig 7 : A simple convnet - LeNet Architecture[4]

It's a simple & relatively standard architecture for a CNN, which is close to the simple above convnet example[5], but in machine learning, it is always a good idea to remember the principle of parsimony.

[4] https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/
[5]

This first CNN will be therefore my **benchmark model.** As our dog breed classification task is complex & the training of CNN require lot of computational resources, it's probable that this first model performs not very well. However, it will be very useful to track the refinement & improvement that I could then do.

To improve the performance of my algorithm, I'll modify the architecture of my CNN to use transfer learning. As explained by Jason Brownlee in his blog which is called "Machine Learning Mastery", transfer learning is a machine learning method where a model developed for a task is reused as the starting point on a second task[6].
I'll create a **new CNN** that can identify dog breed from images and I'll use one of the most efficient models on the image classification task, which is called **Resnet**[7]
I'll try to apply a deeper & more complex architecture for my second CNN based one of Resnet's architectures.
Here below the different Resnet architectures :



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

:ures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of block

Fig 8 : Resnet architectures

I should modify the last layers of Resnet to adapt the architecture to our classification problem with 133 classes.
The performance of this last version of my CNN should perform better & obtain a higher accuracy than my baseline model

C. Benchmark

As explained above, my **benchmark model will be my CNN from scratch**. I'll compare the performance of this baseline model with my second version of CNN, which will be based on Resnet architecture. I expect that my second version of CNN architecture, which will be deeper & more complex, will have a higher **accuracy** on my test dataset.

[6] https://machinelearningmastery.com/transfer-learning-for-deep-learning/
[7] *"Deep Residual Learning for Image Recognition"*, He & al (2015), https://arxiv.org/abs/1512.03385

III. Methodology

A. Data Preprocessing

I mainly used the second dataset, which is composed of 8351 dog images to build my dog breed classification algorithm. As this dataset is composed of dog images that have different characteristics (notably the size), I proceed to several data preprocessing.
I resized & cropped each image to 224x224 pixels.
Moreover, for the sake of performance, I also decided to apply data augmentation & to apply a random horizontal flip on my training data.
Then, I converted each image to a tensor because I wanted to be able to use them in my Pytorch CNN models.
Finally, because of potential differences in the image described above, I also normalized them.

B. Implementation & Refinement

My solution is composed of 2 steps: in a first step, I started to build an algorithm of human face detection with OpenCV & Dlib libraries. My second version of the model from Dlib library performed very well on our human dataset since it successfully detected 100% of human faces in our datasets while reducing the false positive of detected dog face in the dog dataset.
Then I developed a dog detector from a pre-trained model. I used a benchmark model that is called VGG16. VGG16 has been trained on ImageNet, which is a very large dataset used for image classification & other vision tasks. VGG16 model is able to classify an object in a picture in one of its 1000 categories. Thanks to the VGG16 model, my dog detector performs very well because it’s able to detect 91% of dogs in the dog dataset.
To sum up, the goal of this step was to be able to detect if a dog or a human is in an image.

In a second step, I built convolutional neural networks to predict & classify dog breed from images. I started to build a **first convolutional neural network from scratch** based on the following architecture : 3 conv2d layers & 2 linear layers. I decided to apply a stride of 2 on the 2 first conv2D layers to accelerate the training of my model, padding of 1 & also to apply a dropout to limit overfitting. Even with a training of 50 epochs & an adaptive optimizer (Adamax), its performance was low (only 26% of accuracy, that is to say of dog breed correctly classified in %).
I then modified my strategy and I decided to used transfer learning to create a **new CNN** that can identify dog breed from images. I used one of the most efficient models on the image classification task, which is called **Resnet50** (Initially I started to try a more complex benchmark model, Resnet152 but I lacked computational resources in the Udacity’s workspace even when my batch size was small). I only modified the last layers to adapt the architecture to our classification task of 133 classes. As expected, the performance of this last version of my CNN performed very well (86% accuracy), even with a training of 20 epochs.
For the sake of clarity, it’s useful to recall that for the image classification task, one of the metrics frequently used is accuracy.

IV. Results: Performance of my final model vs benchmark model

A. Model Evaluation & Validation

After trained & optimized my 2 models, I obtained model performances above 2 thresholds given by Udacity which were respectively 10% & 60%. This is already the first good result.

B. Justification

Relative to my benchmark model - my CNN built from scratch -, as I expected, my CNN using transfer learning & Resnet50 performs very well. It's a great improvement in the accuracy (88% of accuracy vs only 26%). To note, that the accuracy of my model on the test set is above the required performance threshold which is 60%.

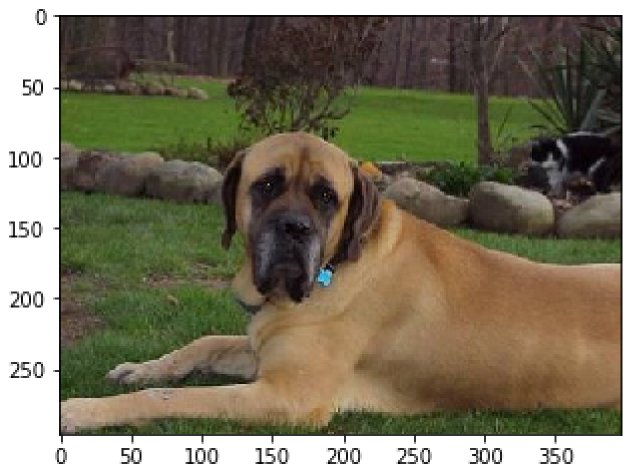
For this kind of image classification task, which is hard, it's a great performance.

V. Conclusion :

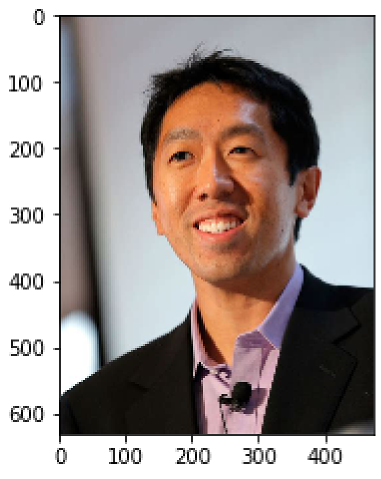
A. Free-Form Visualization

This project has been a great & fun experience for me. For example, when I analyzed my predictions on new images, I found that my results are reasonably significant. I inserted some results below :

Predicted Breed : Mastiff



He ou she looks like a dog of the following breed : Dogue de bordeaux



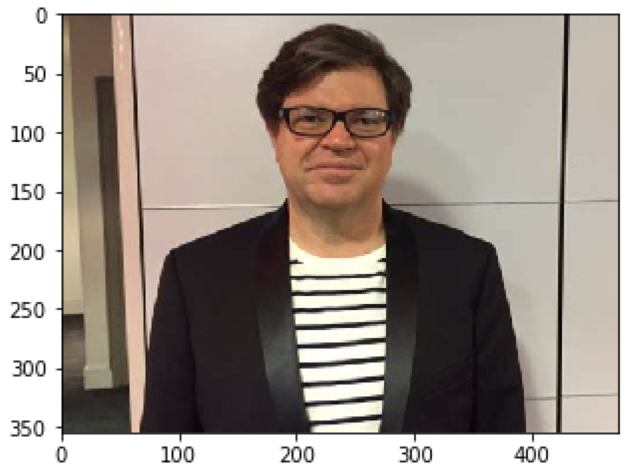
B. Reflection

In this project, I built an app that determines whether an image contains a human, a dog or neither & then provides an estimate of the dog breed (or the dog breed that is the most resembling if a human is detected). This project was very interesting for me because I built a full pipeline to analyze & classify an image. It’s a good project to develop my skills in computer vision.
The building of the dog breed classification was challenging.
The main difficulty in this kind of problem for me was to find the best neural network architecture & to tune the hyperparameters. It’s often a long iteration process. In the present project, I proceed manually to tests different versions of architecture & I finally retained a version that performed well.

C. Improvement

Finally, my last version of the model performs very well on this hard dog breed classification task.

```
He ou she looks like a dog of the following breed : Xoloitzcuintli
```



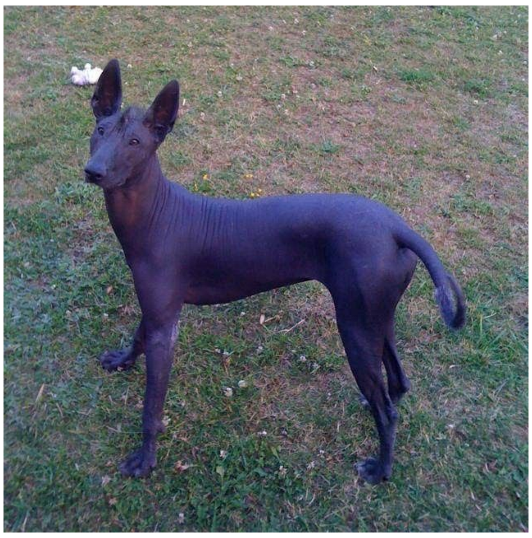


Fig 9 : True Image of Xoloitzcuintli

However, as the above images show, some results seem surprising. So I think there are a lot of points for the improvement of our algorithms.
Firstly, I think we could improve our process with a better image preprocessing. For example with a more complete data augmentation with vertical flipping or rotation, we should easily obtain better results.
Secondly, we could also try to clean better our images. In this notebook, we paid little attention to the quality of our images. What if an image represents a human & a dog or several dogs like the below image extracted from our train dataset.

Thirdly, my CNN architectures don't be very optimized. So with more time to spend to optimize my architecture & to tune my hyperparameters, we'll improve the performance of our classification algorithm. In general, I used the Python library which is called Hyperopt to add an autoML step to optimize my CNN architectures. With Hyperopt I can easily try a lot of hyperparameters range, I can test other optimizers or deeper layers or add more layers.

https://github.com/ChrisBg/Machine-Learning-Engineer-Nanodegree-Udacity/tree/images/my-dog-breed-classifier