Fisheye *doggo*

Arthur Goutallier
arthur.goutallier@student.ecp.fr

Vincent Duault
vincent.duault@student.ecp.fr

1. Introduction

1.1. Context

”The better I get to know men, the more I find myself loving dogs.” Charles De Gaulle certainly knew the true value of a dog, always referred as man’s best friend. Yet there are more than 300 breeds of dogs, from the Chihuahua, to the Saint Bernard, to the beloved Golden Retriever, our favorite one. The internet was not spared the craze for these adorable pets, and it didn’t take long for memes to take over the *doggos*. The goal of our project was to build an algorithm able, taking a dog picture as an input, to automatically produce a fisheye lens focused on his nose.

1.2. Problem definition

First of all, we need to clearly define the kind of *meme* we are talking about. The fisheye lens effect gives a wide angle, the viewer feels like he is watching a panoramic image. To make it funny, we need to have this effect focused on the nose of the dog. The input and output of our algorithm are :

- **Input** An image of a dog, ideally with a plain and light background, to make the dog detection easier. We focused on Golden Retrievers, as they are our favorite breed, and also have a plain color fur.

- **Output** If our algorithm managed to detect the dog’s nose, it should return the same image but distorted by the fisheye effect. Else, it should return an error message.

1.3. Pipeline

Our algorithm is made of two main parts.

- **Nose segmentation** Detection and segmentation of the dog’s nose

- **Fisheye effect** Application of the fisheye effect, taking the nose’s location as its center

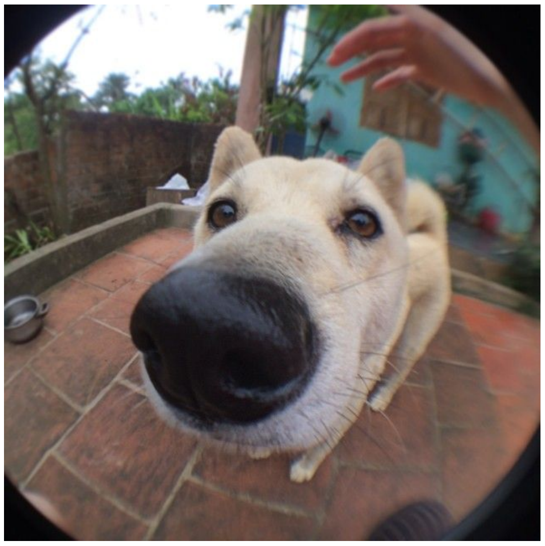


Figure 1: Example of a fisheye effect on a dog’s picture

2. The fisheye effect

2.1. Description

The fisheye effect previously mentioned is inspired by the fisheye lens used in photography. As described on Wikipedia [4], a fisheye lens is an ultra wide-angle lens that produces strong visual distortion intended to create a wide panoramic or hemispherical image.Fisheye lenses achieve extremely wide angles of view. Instead of producing images with straight lines of perspective, fisheye lenses use a special mapping , which gives images a characteristic convex non-rectilinear appearance.

2.2. Implementation

The fisheye lenses are based on a barrel distortion. In this kind of distortion, the image magnification decreases with distance from focal axis, giving the impression that the image has been mapped around a barrel (or a sphere). The barrel distortion model is :

$$r_u = r_d(1 + k{r_d}^2) \quad (1)$$

where r_u and r_d are the distance from the centre of distortion in the non distorted and distorted images, and k is the distortion parameter, see Figure 2.

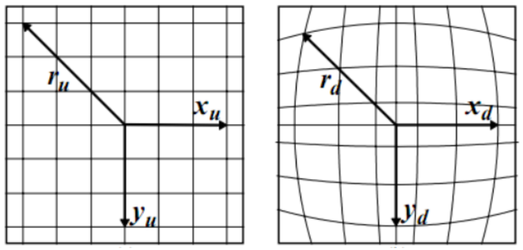
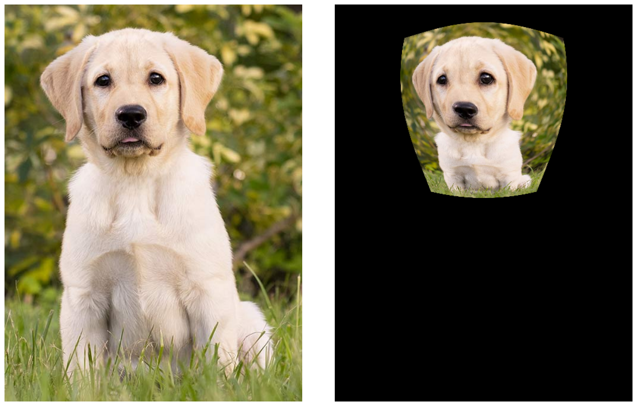


Figure 2: Barrel distortion

In order to implement this transformation, we used *OpenCV* geometric image transformations functions. One of these is designed to compensate for lens distortion, a little twist allowed us to perform the desired transformation. Besides the previous parameters, we also had to set the camera center, the focus point for our fisheye effect, and the focal length.



(a) Input image (b) Output image

Figure 3: Example of our barrel transformation algorithm, where $k = 10^{-3}$, the focal length $f = 5$, centered on the dog's left eye

3. Nose detection

The next task aims to find the location of the dog's nose on the picture. We assumed that the center of a bounding box surrounding the nose would be a reasonable point to apply the fisheye effect on.

3.1. First considerations

Considering the fact that we are focusing on Golden Retriever or similar breeds, we noticed that the nose was one of the only dark parts of the dog, along with his eyes. Therefore we thought of an algorithm that could spot this ”triangle” made by three dark regions of the dog's face.

3.2. Superpixel approach

A superpixel is a group of pixels that share some common characteristics. The idea is to use this method to segment the image in light or dark regions, and then spot the ones we are interested in, the nose and the eyes. SLIC (Simple Linear Iterative Clustering) [1] is an algorithm that generates superpixels by clustering pixels based on the color similarity and spatial proximity in a 5 dimensions space :

- **x, y** the coordinates
- **L, a, b** the CIELAB colorspace

SLIC takes as an input the number of approximately equally-sized superpixels. We chose this algorithm for our baseline for its speed and its proven efficiency.

We first run the *scikit-image* implementation of SLIC on an input image, previously converted to gray scale and eventually blurred. Then we set each superpixel's value to the mean of all its pixels values. We can order the superpixels by descending darkness, and select an arbitrary subset of them, as show on Figure 4.

We studied the influence of the blurring and the number of superpixels on the ability of our algorithm to find the nose region. In Figure 5, when plotting the 10 darkest regions, only 5d, 5e, 5f and 5g succeeded in spotting the eyes besides the nose.

3.3. Finding the nose

Now that we are able to spot the dark regions of any dog picture, we need to be able to determine which one of them is the nose. We set two criteria :

- **Isocele criteria** The nose region must be a equidistant to two other dark regions, which would supposedly be the eyes.
- **Eyes alignments criteria** The eyes region candidates selected previously must be almost on the same horizontal line. Our main assumption is that on every picture, the dog is straight and right side up.

At first we only implemented the first criteria, but the background would distort the results, creating failed fisheye effects as show in Figure 6. Indeed some dark background spots can be detected, and can sometimes create isocele triangles. The second criteria allowed us to filter out some of the false positives, but the background remains one of our main issues.

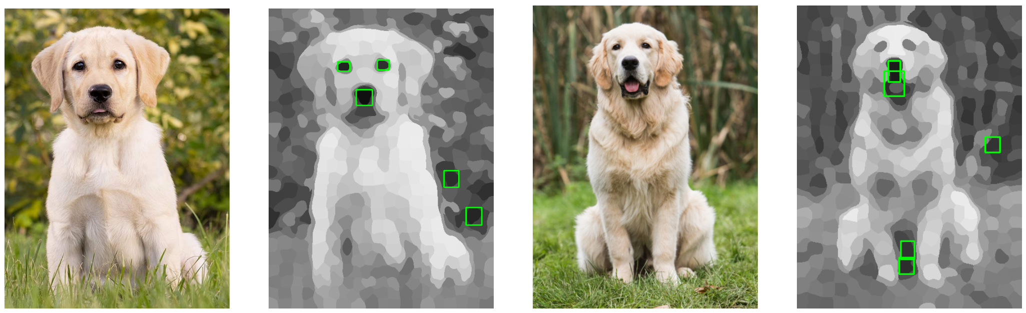

(a) First input image (b) First output image (c) Second input image (d) Second output image

Figure 4: SLIC algorithm ran on two different images, with 500 superpixels and a Gaussian smoothing kernel size of 5. The green bounding boxes surround the 5 darkest superpixels

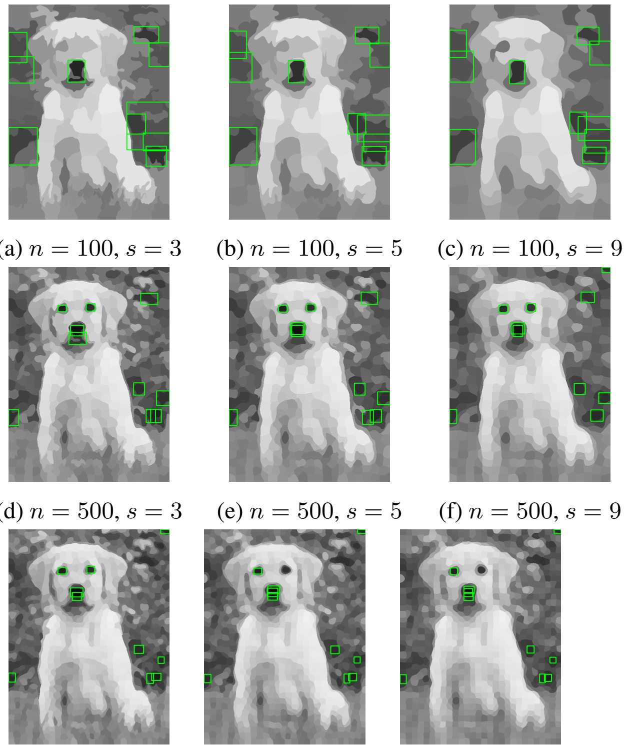

(a) $n = 100$, $s = 3$ (b) $n = 100$, $s = 5$ (c) $n = 100$, $s = 9$ (d) $n = 500$, $s = 3$ (e) $n = 500$, $s = 5$ (f) $n = 500$, $s = 9$ (g) $n = 1000$, $s = 3$ (h) $n = 1000$, $s = 5$ (i) $n = 1000$, $s = 9$

Figure 5: Testing the influence of the number of superpixels n and the Gaussian smoothing kernel size s. The green bounding boxes surround the 5 darkest superpixels.

4. Dog Face Recognition

the triangle between the two eyes and the nose is to first find the head of the dog. As such, we want to use a classifier capable of detecting the faces of the dogs on the images we provide.

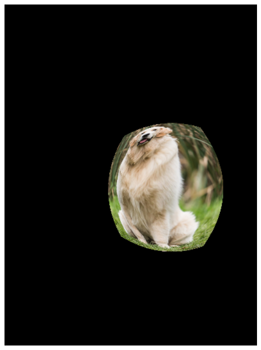


Figure 6: Output with fisheye effect, isocele criteria met and eyes alignement criteria not met

4.1. Cascade Classifier

To stick with the idea of the course, we wanted to train a Cascade Classifier. The purpose of such a classifier is to apply filters on an image, one after another, and if all filters are deemed passed, the region of the photo is labelled as having a dog face. However, due to our peculiar angle in this project, finding a Cascade Classifier already pre-trained on dog faces is not an easy task. Furthermore, it was also a good way to learn how to train one, so we decided to train from scratch using openCV.

Although we rapidly faced a problem : how do we get our hands on a labelled dataset.

4.1.1 Using CNN for labeling data set

Even though we don't want our final code to have a deep-learning tool in it, we decided it would be quite handy to label our data set using a pre-trained cnn on dog faces [3].

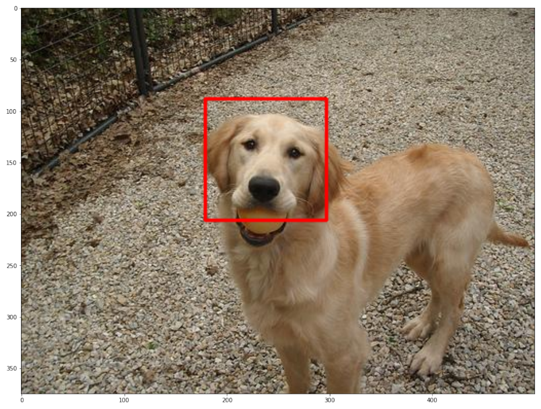


Figure 7: CNN face detection

4.2. Training Cascade Classifier

Once we had the training set labelled, we also needed a negative data set, fortunately unlabelled. We chose a set of background from Stanford data set library.

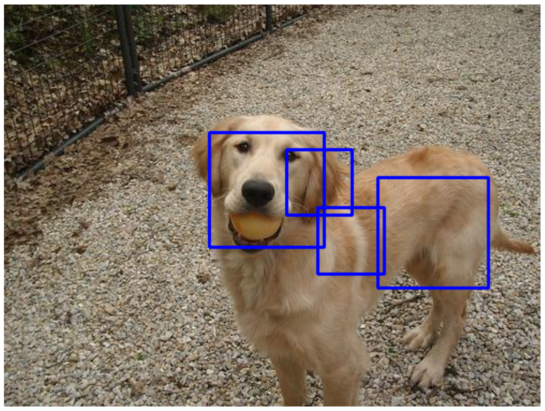


Figure 8: Cascade face detection

4.3. Results on Cascade Classifier

Considering these results, the pipeline is quite simple :

- Face Detection
- Nose Detection
- Fisheye Effect

Regarding the results between cascade face detection and CNN face detection, we deemed logical to compare for each area located from cascade the surface from CNN covered.

Surface Coverage	Number of Photos
0	165
0.1	101
0.2	91
0.3	76
0.4	66
0.5	72
0.6	56
0.7	41
0.8	21
0.9	5
Average	0.28
Average without zeros	0.37

[2]

All things considered, the results seem acceptable for the first part of the filtering. We should understand that the dataset is quite different since the dog is not always the same colour, in the same spot, or looking the same way.

References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2274–2282, 2012. 2

[2] K. G. . C. J. . D. Bailey. A real-time fpga implementation of a barrel distortion correction algorithm with bilinear interpolation. 2003. 4

[3] kairess. `https://github.com/kairess/dog_face_detector/blob/master/main.ipynb`. 3

[4] Wikipedia. Fisheye lens. `https://en.wikipedia.org/wiki/Fisheye_lens`. 1