Lab 1 - Computer Vision

Diego Torres

November 2023

1 Introduction

In this lab exercise we aim at using the Bag-of-Features (BoF) method, to encode a dataset of images of Rubik's cubes. In NLP, a Bag-of-Words model tries to represent a document as the counts of its composing terms. Although simple in practice, extensions of this idea can lead to powerful methods like TF-IDF, which work very well in information retrieval tasks. Coming back to computer vision, the main difficulty of applying this method to images is to come up with an appropriate set of "terms" to extract from an image. This is where Harris' corner detection algorithm and Histogram of Oriented Gradients (HOG) come into place.

In the following sections I will explain the steps that I did to encode the images with the BoF method. I will introduce the main ideas and theories behind the algorithms, and provide the code that was used.

2 Harris' Corner Detection

The intuition behind this algorithm is that corner points can be regarded as points where the image intensity changes in multiple directions. This can be seen as a generalization of detecting edges as points of high intensity change in a single direction, large gradients. Therefore, gradients will also play a big role in this algorithm.

The core step in Harris' algorithm is to use the gradient components I_x and I_y to build, at each pixel, the matrix M:

$$M = \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}. \tag{1}$$

We then compute $R = \det M - k(\operatorname{Tr} M)^2$ (with k being between 0.04 and 0.06). According to the algorithm, the corner points will be the local maxima of this value.

The implementation that I did has additional steps designed to make it more robust.

1. Apply gaussian filter on the image
2. Compute the derivatives and apply a gaussian filter on them
3. Compute the matrices M and R
4. Set the entries of R that are smaller than $\text{threshold}\max(R)$ to 0.
5. For the remaining entries, keep only the maxima in sliding windows of fixed size.

Hence, our algorithm will have several hyper parameters, the two variances of the gaussian filters, the threshold of the entries of R, and the size of the sliding window. After a short period of experimentation, I found that 1, 0.6, 0.01 and 5 worked well with the dataset. The Python code to perform this algorithm is available in Code 1, and a working example is shown in figure 1.

```python
class HarrisDetector:
    img_sigma = 1
    grad_sigma = 0.6
    threshold = 0.01
    ws = 5

    def __init__(self):
        self.R = None

    def harris_detection(self, image):
        filtered_image = gaussian_filter(image, sigma=self.img_sigma)
        Ix, Iy = compute_gradients(filtered_image)
        Ixx = Ix**2
        Iyy = Iy**2
        Ixy = Ix*Iy
        Ixx = gaussian_filter(Ixx, sigma=self.grad_sigma)
        Iyy = gaussian_filter(Iyy, sigma=self.grad_sigma)
        Ixy = gaussian_filter(Ixy, sigma=self.grad_sigma)
        det = Ixx*Iyy - Ixy**2
        trace = Ixx + Iyy
        R = det - 0.04*trace**2
        R = self.non_max_supression(R)

        self.R = R

        return self

    def non_max_supression(self, R):
        xx, yy = np.where(R > self.threshold*R.max())
        R = np.where(R > self.threshold*R.max(), R, 0)

        for x, y in zip(xx, yy):
            if x < self.ws or y < self.ws or x > R.shape[0]-self.ws or y > R.shape[1]-self.ws:
                R[x, y] = 0
                continue
            values = R[x-self.ws:x+self.ws, y-self.ws:y+self.ws]
            R[x-self.ws:x+self.ws, y-self.ws:y +
                self.ws] = np.where(values == values.max(), values, 0)

        return R

    def draw_corners(self, image):
        plot_image = np.expand_dims(image, 2).repeat(3, 2)
        for x, y in zip(*self.get_corners()):
            cv2.circle(plot_image, (y, x), int(self.threshold *
                       max(plot_image.shape)), (255, 0, 0), -1)

        return plot_image

    def get_corners(self):
        return np.where(self.R > self.threshold*self.R.max())
```

Code 1: My Python implementation of Harris' algorithm

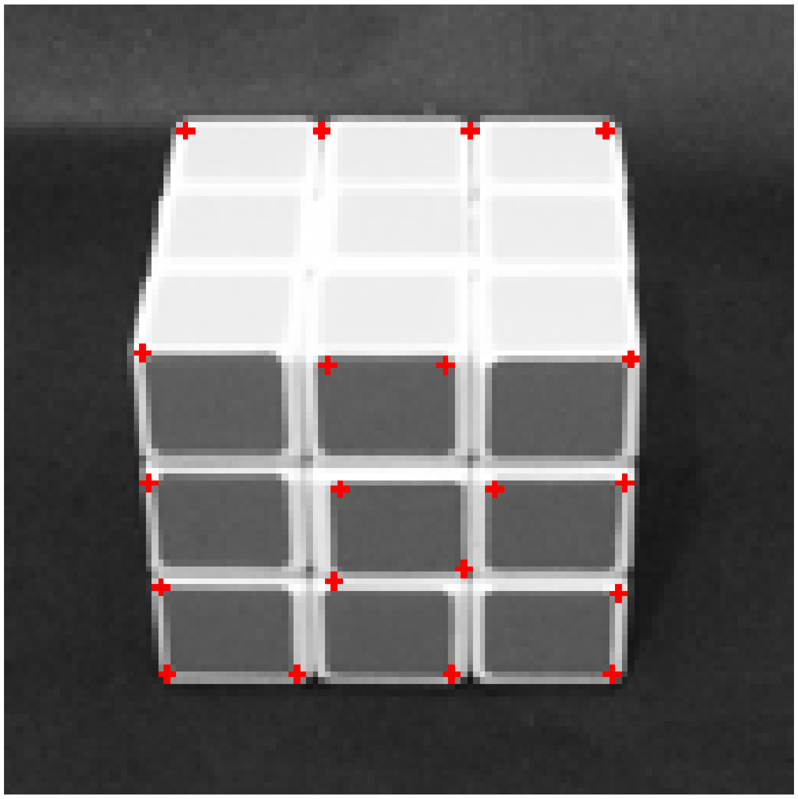

Figure 1: Implementation of Harris' on one image from the dataset

```python
def orientation_binning(magnitude, orientation, cell_size, num_bins=4):
    bin_edges = np.linspace(0, 360, num_bins+1, endpoint=True)
    cell_rows, cell_cols = magnitude.shape[0] // cell_size[0], magnitude.shape[1] // cell_size[1]
    hog_vector = np.zeros((cell_rows, cell_cols, num_bins))

    for i in range(cell_rows):
        for j in range(cell_cols):
            cell_magnitude = magnitude[i*cell_size[0]
                :(i+1)*cell_size[0], j*cell_size[1]:(j+1)*cell_size[1]]
            cell_orientation = orientation[i*cell_size[0]
                :(i+1)*cell_size[0], j*cell_size[1]:(j+1)*cell_size[1]]

            for b in range(num_bins):
                bin_mask = (cell_orientation >= bin_edges[b]) & (
                    cell_orientation < bin_edges[b+1])
                hog_vector[i, j, b] = np.sum(cell_magnitude[bin_mask])

    hog_vector = hog_vector.ravel()

    norm = np.linalg.norm(hog_vector)
    if norm != 0:
        hog_vector /= norm

    return hog_vector


def hog_descriptor(image, cell_size=(8, 8), num_bins=4):
    grad_x, grad_y = compute_gradients(image)
    magnitude = np.sqrt(grad_x ** 2 + grad_y ** 2)
    orientation = np.arctan2(grad_y, grad_x) * (180 / np.pi)
    orientation = np.mod(orientation, 360)
    hog_vector = orientation_binning(
        magnitude, orientation, cell_size, num_bins)
    return hog_vector
```

Code 2: Python code for HOG

3 Histogram of Oriented Gradients

The Histogram of Oriented Gradients (HOG) is a feature descriptor used in computer vision and image processing for the purpose of object detection. The main idea is to build a histogram of the gradients that captures how fast the image changes in certain directions. The gradient of each pixel casts a vote on this histogram proportional to its magnitude, and the bin this vote goes to depends on the value of its orientation. This process is done over small patches in the image, so that its full representation is given by the concatenation of these smaller vectors.

A more detailed description of the algorithm I used is:

1. Compute the gradients of the image
2. Get the magnitude and orientation of the gradients
3. Split the image in cells of fixed size and for each cell:
   (a) Create a vector of zeros with size equal to the number of bins of the histogram
   (b) Fill each entry in this vector as the sum of the magnitudes of the gradients whose direction falls within the bin
4. Concatenate all vectors and normalize it with respect to the L2 norm

The Python implementation of this algorithm is available in Code 2.

```
def extract_subimage(img, center_ix, size):
    x = int(center_ix[0] - size/2)
    y = int(center_ix[1] - size/2)
    subimage = img[x:x+size, y:y+size]

    return subimage

def extract_image_features(image):
    harris = HarrisDetector().harris_detection(image)
    corner_xx, corner_yy = harris.get_corners()

    image_features = []
    for x, y in zip(corner_xx, corner_yy):
        subimage = extract_subimage(img, (x, y), 16)
        hog_features = hog_descriptor(subimage, cell_size=(8, 8), num_bins=4)
        image_features.append(hog_features.reshape(1, -1))
    image_features = np.concatenate([arr for arr in image_features if arr.shape == (1, 16)], axis=0)
    return image_features

all_features = []
for dic in data:
    image = dic['img']
    image_features = extract_image_features(image)
    dic['features'] = image_features
    all_features.append(image_features)

all_features = np.concatenate(all_features, axis=0)
model = KMeans(n_clusters=6, random_state=0).fit(all_features)

for dic in data:
    clusters = model.predict(dic['features'])
    dic['hist'] = [0]*6
    for cluster in clusters:
        dic['hist'][cluster] += 1
```

Code 3: Python code for BoF

4 Bag-of-Features

Armed with these two algorithms, we can extract a meaningful set of features in each image by combining them in the following way. First, extract the corner points, we consider them as indicators of important areas in the image. Then, compute the HOG features of a neighboring region of each corner point. After these two steps, each image will be associated with a set of vectors, each of which of the same size.

The result of this procedure still does not look like a Bag-of-Features model, as these vectors cannot be interpreted as defined terms. In order to do that we use the k-means algorithm on the full dataset of image features. This allows us to replace each feature vector of the image by the cluster it belongs to. Therefore, the terms of an image will be the clusters in which its feature vectors were assigned. Finally, count the number of times a cluster is present in the image to get a quantized vector representation.

In my implementation I use the square region of size 16 with center at the feature as its neighboring region to extract the HOG features. I use a cell size of 8x8 and I split the unit circle in 4 bins (0-90, 90-180, 180-270, 270-360 degrees). Additionally, I used 6 clusters with k-means. The Python code is available in Code 3.

5 Results

In figure 2, I show the entirety of the dataset, marking the extracted corners using Harris' algorithm and showing the quantized vectors in the title.

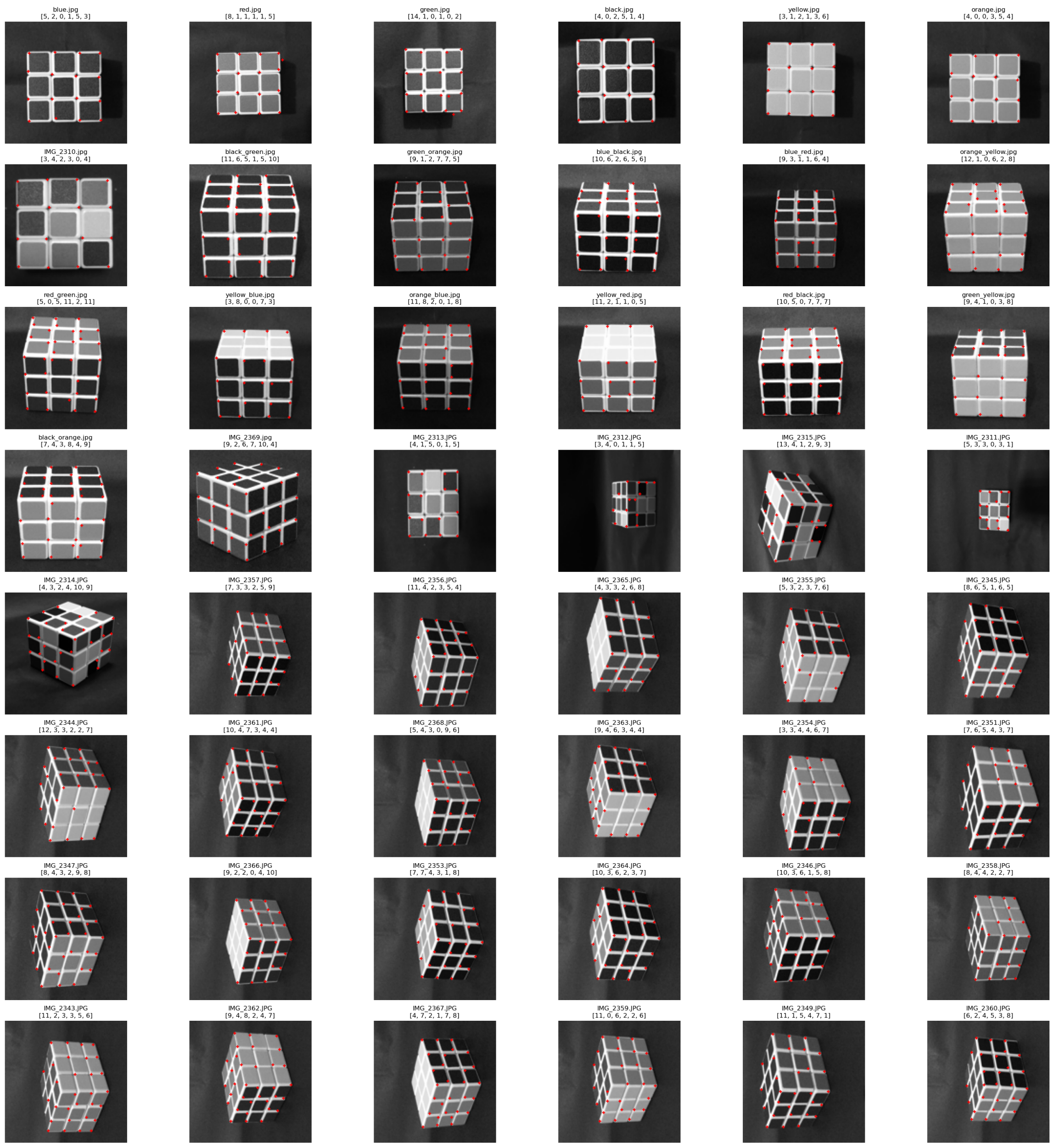

Figure 2: BoF applied to the entire dataset