FEATURE EXTRACTION FROM IMAGE

Pixel Average-Median-Mode &

Edge Features

Kshitij Gour (0801CS161035)

Sanchit Bhat (0801CS161075)

Contents

1. Introduction

1.1. Pixel Average-Median-Mode

A digital photo is not one non-dividable thing. If we zoom in far enough, we'll see that our image is like a mosaic formed by small square tiles, which are called pixels. We see images as they are – in their visual form. Machines, on the other hand, store images in the form of numbers. Each number corresponds to a pixel value which denotes the intensity or brightness of the pixel. Smaller numbers (closer to zero) represent black, and larger numbers (closer to 255) denote white. A colour image is typically composed of multiple colours and almost all colours can be generated from three primary colours – red, green and blue. Hence, in the case of a colour image, there are three Matrices (or channels) – Red, Green, and Blue. Each matrix has values between 0-255 representing the intensity of the colour for that pixel. To calculate Mean, Median, Mode of these pixel values for a colourful image, we need to perform these functions on each channel separately to get these values for each channel.

1.2. Edge Features

Nowadays, Digital image processing is an ever expanding and dynamic area with applications reaching out into our everyday life such as in medicine, space exploration, surveillance, authentication, automated industry inspection and in many more areas. For building such high-speed systems, Edge Detection and Feature-Extraction, as an important process in motion recognition is required to be processed. Edge is a basic feature of an image. Edges define the boundaries between regions in an image by locating sharp discontinuities in pixel values, which helps with segmentation and object recognition. Image Edge detection significantly reduces the amount of data and filters out useless information, while preserving the important structural properties in an image. Since edge detection is in the forefront of image processing for object detection, this project describes Canny Edge Detection algorithm for extracting edge features of an image.

2. Mathematical Formulation

2.1. Formulation

Canny edge detection is a multi-step algorithm that can detect edges with noise suppressed at the same time.

1. Smooth the image with a Gaussian filter to reduce noise and unwanted details and textures.

$$g(m,n) = G_\sigma(m,n) * f(m,n)$$

   where,

$$G_\sigma = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{m^2+n^2}{2\sigma^2}\right)$$

2. Compute gradient by using any of the gradient operations (Roberts, Sobel, Prewitt, etc.) to get:

$$M(n,n) = \sqrt{g_m^2(m,n) + g_n^2(m,n)}$$

   and

$$\theta(m,n) = tan^{-1}[g_n(m,n)/g_m(m,n)]$$

3. Threshold M:

$$M_T(m,n) = \begin{cases} M(m,n) & \text{if } M(m,n) > T \\ 0 & \text{otherwise} \end{cases}$$

   Where, T is so chosen that all edge elements are kept while most of the noise is suppressed.

4. Suppress non-maxima pixels in the edges in M_T obtained above to thin the edge ridges (as the edges might have been broadened in step 1). To do so, check to see whether each non-zero $M_T(m, n)$ is greater than its two neighbours along the gradient direction $\theta(m,n)$. If so, keep $M_T(m, n)$ unchanged, otherwise, set it to 0.

5. Threshold the previous result by two different thresholds τ_1 and τ_2 (where $\tau_1 < \tau_2$) to obtain two binary images T_1 and T_2. Note that T_2 with greater τ_2 has less noise and fewer false edges but greater gaps between edge segments, when compared to T_1 with smaller τ_1.

6. Link edge segments in T_2 to form continuous edges. To do so, trace each segment in T_2 to its end and then search its neighbours in T_1 to find any edge segment in T_1 to bridge the gap until reaching another edge segment in T_2.

3. Algorithm

3.1. Canny Edge Detection Algorithm

Algorithm:

Input: 3-dimensional image, lower threshold (optional), upper threshold (optional)

Output: 2D matrix containing extracted edge features

1. Image is received as a three-dimensional matrix of shape M x N x 3.
2. Find the median pixel value from the matrix.
3. If lower and upper threshold is None, calculate lower and upper threshold values using following formula:
   3.1. Lower bound: max (0, 0.7 * median value)
   3.2. Upper bound: min (255, 1.3 * median value)
4. Blur the image using a kernel of size (5, 5).
5. Use blurred image as input along with lower bound and upper bound in cv2.Canny()
6. Store the extracted features in an array.
7. Display original image and extracted edge features side by side.

4. Documentation of API

4.1. Package Organization

from pixel_features import pixel_features

from Edge_Features import Edge_Features

__init__() not defined in both classes.

4.2. Methods

def edge_detection_canny(image, lower = None, upper = None)

Returns edge features of an image using canny edge detection algorithm

Parameters:

image: input three-dimensional image

lower: lower threshold value for canny edge detection (optional)

upper: upper threshold value for canny edge detection (optional)

def horizontal_edge_prewitt (image)

Returns horizontal edge features of an image using prewitt_h

Parameters:

image: input two-dimensional grayscale image

def vertical_edge_prewitt (image)

Returns vertical edge features of an image using prewitt_v

Parameters:

image: input two-dimensional grayscale image

def mean_pixel_value (image)

Returns numpy array containing average pixel values of each channel and entire image

Parameters:

image: input three-dimensional image

def median_pixel_value (image)

Returns numpy array containing median pixel values of each channel and entire image

Parameters:

image: input three-dimensional image

def mode_pixel_value (image)

Returns numpy array containing most frequent pixel values of each channel and entire image

Parameters:

image: input three-dimensional image

5. Examples

5.1. Example 1

image = cv2.imread(‘road_image.jpg’)

image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

edges = features.edge_detection(image)

Output:array([[0, 0, 0, ..., 0, 0, 0],

[0, 0, 0, ..., 0, 0, 0],

[0, 0, 0, ..., 0, 0, 0],

...,

[0, 0, 0, ..., 0, 0, 0],

[0, 0, 0, ..., 0, 0, 0],

[0, 0, 0, ..., 0, 0, 0]], dtype=uint8)

5.2. Example 2

image = cv2.imread (‘flat_chessboard.png’, cv2.IMREAD_GRAYSCALE)

edges = features.horizontal_edge_prewitt (image)

Output:array ([[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,

0.00000000e+00, 0.00000000e+00, 0.00000000e+00],

[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,

-5.55111512e-17, -5.55111512e-17, 0.00000000e+00],

...,

[0.00000000e+00, -5.55111512e-17, -5.55111512e-17, ...,

0.00000000e+00, 0.00000000e+00, 0.00000000e+00],

[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,

0.00000000e+00, 0.00000000e+00, 0.00000000e+00]])

5.3.Example 3

image = cv2.imread('puppy.jpg')

median = features.median_pixel_value(image)

Output:array([23., 32., 26., 27.])

5.4.Example 4

image = cv2.imread ('puppy.jpg')

mean = features.mean_pixel_value (image)

Output:array([43.78883905, 51.82413609, 57.27795148, 50.96364221])

6. Learning Outcomes

- Capacity to integrate theoretical and practical knowledge to evaluate features of an image using different mathematical functions.
- Extracted and analysed some important features related to pixels and edges of an image.
- Analysed the way in which machine stores and processes images in the form of arrays.
- Analysed and evaluated higher mathematical concepts with the ability to clearly implement and present the conclusions and the knowledge behind it.
- Capacity to design and perform research on different aspects of feature extraction from images. It can help with real life applications in form of computer vision projects.

Appendix A

References

1. https://www.analyticsvidhya.com/blog/2019/08/3-techniques-extract-features-from-image-data-machine-learning-python/?#

2. https://www.mygreatlearning.com/blog/feature-extraction-in-image-processing/

3. http://fourier.eng.hmc.edu/e161/lectures/canny/node1.html

4. https://automaticaddison.com/how-the-canny-edge-detector-works/

5. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html#canny