Alireza Mohammadshafie – Exam 1 CSCE 5222 Feature Engineering

1-To represent 88 distinct gray values, you need enough bits per pixel, to calculate that number we have to know that each extra bit roughly doubles how many colors we can represent. 6 bits gives us 64 different colors(2^6 = 64). And with 7 bits we can represent 128 different colors(2^7 = 128) which is enough for our work, So the minimum color depth is 7 bits per pixel for this image.

2- The target histogram is approximately uniform across the intensify range and we will use this method to spread pixel intensities evenly, maximizing contrast and using the full dynamic range(by making the cumulative distribution function(CDF) of the output approximately linear). In practice, it's only approximately uniform duo to discrete levels and image content.

3- HOG feature extraction works by first computing pixel-wise gradients to get gradient magnitudes and orientations that highlight edges. The image is split into small cells(like 8*8 pixels), and each cell builds a histogram of edge orientations where each pixel votes to bins (commonly 9 bins over 0-180) weighted by its gradient magnitude, often with interpolation across neighboring bins/cells to reduce aliasing. Cells are then grouped into overlapping blocks(2*2 cells), their histograms concatenated and normalized to achive robustness to illumination and contrast changes. Finally, all normalized block vectors across the image are concatenated into a single fixed-length descriptor. In short: gradients capture edges, cell histograms summarize local shape, block normalization provides lighting invariance, and concatenation yields a robust, spatially aware feature vector for classifiers.

4- HSV color space. The intensity is the V (value) channel.

5- first we have reverse the filter, So it will be like [-2 , 0, 1]
then we should multiply it step by step to the signal. (It is easier to show the convolution with the histogram of those)
y[0] = 4 = 1*4
y[1] = 9 = 1*9 + 2*4
y[2] = -6 = (-2 * 4) + 0*9 + 1*2
y[3] = -18 = 1* 0 + (-2 *9) + 0*2
y[4] = 1 = 0*0 + 1*5 +(-2* 2)

y[5]=5 = (-2 * 0) + 0*5 + 1*5

y[6] = -10 = (-2*5) +0*5 +0*1

y[7] = -10 = (-2*5)
y[8] = 0 = (-2*0)

y= [4 9 -6 -18 1 5 -10 -10 0]

6- odd-sized kernels have a single center pixel, so you can align the filter exactly with the target pixel. This gives a symmetric neighborhood with equal pixels on all sides and avoids half- pixel shifts. It also makes same padding straightforward and preseves image size cleanly. Many common symmetric filters (e.g, Gaussian, sabel) are naturally defined around a central element.

7-[[-1 0 1]
[-2 0 2]
[-1 0 1]]

It computes the horizontal derivative (left negative, right positive) to detect verical intensity changes. In addition, The vertical weights [1 2 1] smooth along the edge direction, reducing noise. And also antisymmetry captures change (not brightness) and the 1-2-1 pattern approximates gentle Gaussian smoothing.

8- 1- Gaussian smoothing: blur to reduce noise.
2- Gradient computation: use sobel to get Gx and Gy then magnitude and direction.
3- Non-maximum suppression: Thin edges by keeping only local maxima along the gradient direction.
4- Double thresholding: Label strong (high), weak (low) and non-edge (below low).
5- Hysteresis: keep weak edges connected to strong ones; discard the rest.

9- 1/9 * [1 1 1 1 1 1 1 1 1]

above 3*3 matrix can replace each pixel with the average of its 3*3 neighborhood, blending nearby values and reducing noise/ high-frequency detail. And we can use it for quick denoising, preprocessing before edge detection/ segmentation. And anti-aliasing when downsampling.

10- Harris corner response compute image gradients and then form the second moment matrix over a local window. And then with some formulas it will compute corner score for every pixel(R). large positive R means both underlying eigenvalues are large So that point is a corner. Negative R indicates edges and small R show us the flat points.

11- Shi-Tomasi keeps the same gradient matrix as Harris but changes the selection rule. It scores each pixel by the smaller of the two eigenvalues and declares a corner if that value exceeds a threshold. Harris instead uses a determinant_trace proxy with a tunable parameter to score pixels. In short, Shi_Tomasi directly checks the weakest direction(no tuning), while Harris relies on a weighted proxy that needs a parameter choice.