# Input: image of “gradient world” (artificial, only patches exist)

# Output: Identification of patches and labeling

Added Parameters

1. **include** = control how many pixels to participate while making the segments. As this is percent from total image pixels the values are between 0 and 1

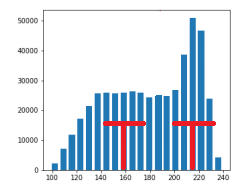
2. **range\_of\_labels**  = control the value range of the pixels, we classify pixels by gradient direction, we used 360 as this is the common way to show degrees (larger number may yield finer results)

3. **window\_size** = to group pixels ,we used a fixed group size. This makes each group to “hold” all the pixels in window\_size degrees to the left and right (in the first iteration).

Algorithm

1. Pre-Process the image – each pixel will now represent the direction of the gradient in said pixel (we used Sobel to find dx , dy )

2. Group new pixels – using np.unique we can make an array that his indexes the represents the direction in degrees, and his values that represents how many pixels point in the direction. Like below



Now we accumulate top groups, saving for each group its median direction to represent it, once we group enough pixels, we return the list of medians. ([160,210] in example)

3. Initial guess – each group has a label, we take the processed image , and for each pixel we assign high probability to the label he is most close to (to group’s median value).

4. while true

4.1 Update – for each pixel, for each label, update his probability using the support of **nearby** pixels (same label attract each other)

4.2 calculate difference in local average consistency ( = it’s a weighted sum of all pixels and theirs labels ), break if diff smaller than epsilon

5. return image, when each pixel is assigned to the label with highest probability

The components for the relaxation labeling network are

1. objects = the pixel, defined by (x,y)

2. labels = the world can be described as a group of constant gradient patches, each patch is assigned different label

3. the initial confidence = we can be somewhat certain that a pixel is belong to his nearest patch (\*)

4. compatibility function = how 2 different pixels react to each other, considering their labels. in our implementation only immediate neighbor with same label support each other.

Observations

1. the edges of the patches make an abrupt change color, which interfere the gradient calculation, one possible solution is to use the gradient magnitude to denote those edges (and some edge detector) , and ease out their values when we group pixels (“don’t make groups for those pixels values”)

2. In the preprocessing ,when the patches that are vertical, there is more noise ,we think it might happen because the values of a steep slope are high and that make it sensitive to noise. One possible solution is to look at polar coordinates