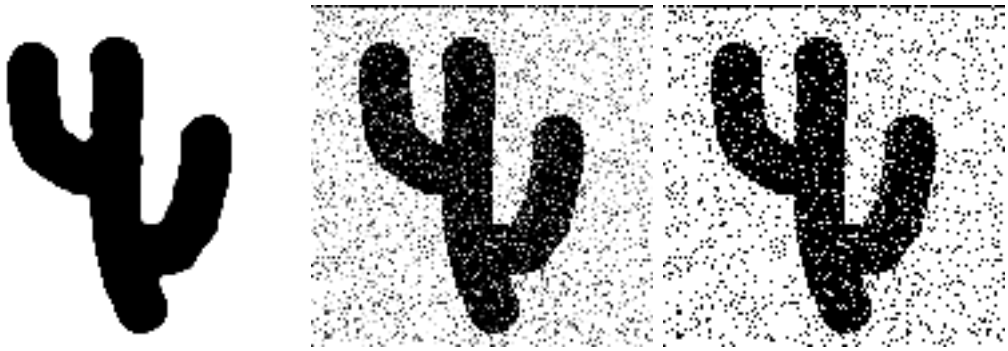
Due: Dec 7, 2018 **Homework 6** CPSC 475

1 In this assignment we will construct a simple relaxation labeling network to label pixels of monochrome images into two classes: *object* pixels and *background* pixels. The world for which we want to apply this system is pretty simple — objects tend to be dark (i.e., project to low intensity pixels), while the backgrounds tend to be bright (i.e., project to high intensity pixels). If the imaging process of our world was perfect, a simple thresholding could have established the required segmentation. However, the cameras with which we image our world are flawed, and introduce a relatively high level of noise. Consequently, the classification based on thresholding results in a very noisy segmentation:



(a) Ideal (b) Noisy (c) Thresholded

Figure 1: While ideal images of objects in our world should look like the one on the left, the image process results in noisy images (middle), which yields noisy segmentation based on thresholding (right, black=*object*, white=*background*). Your mission is to achieve better segmentation using relaxation labeling.

Design a relaxation network that achieves a better segmentation. Here you will use the continuous version of the algorithm, found in section 26.5 in the lecture notes (but make sure you also understand the discrete version, i.e. the label-discarding algorithm discussed in class). Recall that a relaxation labeling network consists of a set of *nodes*, a set of *labels* that can be assigned to those nodes, and a *compatibility function* specifying the compatibility between an assignment of a label at one node, and an assignment of a label at a nearby node. An update rule calculates support for each label at each node using the compatibility function and the assignment of labels nearby, and increases or decreases the *confidence* of the label at that node based on the amount of support it receives from its neighborhood.

Making things more formal, here are a few equations you'll need to use. The support for a

particular label λ at a particular node i is calculated by:

$$s_i(\lambda) = \sum_{j \in N(i)} \sum_{\lambda' \in \Lambda} r_{ij}(\lambda, \lambda') p_j(\lambda'),$$

where $N(i)$ is the set of nodes in the neighborhood of node i, and Λ is the set of labels a node can be assigned. $r_{ij}(\lambda, \lambda')$ specifies the compatibility between a label λ at node i and a label λ' at node j. Note that compatibilities should be symmetric ($r_{ij}(\lambda, \lambda') = r_{ji}(\lambda', \lambda)$), and that compatibilities can be *negative* if necessary! In other words, since there are only two possible labels for this problem, you must decide on a scalar value for each neighboring position in two possible cases: $\lambda = \lambda'$ (compatible) and $\lambda \neq \lambda'$ (incompatible).

$p_j(\lambda')$ is the confidence of the label λ' at node j. Confidences must lie in the range $[0, 1]$, and $\sum_{\lambda \in \Lambda} p_i(\lambda)$ must be 1. In this problem, you only need two labels — given the constraints just mentioned, do you need to keep explicit track of confidence values for both labels, or can things be simplified?

Once you know the support, update the confidence of a particular label λ at node i using the following update rule:

$$p_i^{t+1}(\lambda) = \Pi_0^1[p_i^t(\lambda) + \delta s_i^t(\lambda)],$$

where the operator Π_0^1 truncates its argument to lie in the range $[0, 1]$ (e.g., it sends -0.3 to 0, 1.2 to 1, and leaves 0.2 alone), and δ is a scalar (in general, you want this fairly small, but in this assignment you can use $\delta = 1$). Be sure that after this step, $\sum_{\lambda \in \Lambda} p_i(\lambda) = 1$.

This procedure of calculating support and updating confidences should be iterated until you get a clean image.

In designing your network, think in particular about the following points:

- What are the nodes?
- What are the labels? (Don't use more than two.)
- How should the initial confidences $p_i^0(\lambda)$ be set?
- What should the neighborhood relationships be?
- How should the compatibility fields $r_{ij}(\lambda, \lambda')$ be structured?

Implement your network as a MATLAB or Python program and run it on the `cactus`, `parts`, and `bump` images, all of which contain a clear set of objects on a background. **Do not use blurring or other noise removal techniques!** Submit a short discussion on the above points, your code, and images that illustrate the segmentation based on different numbers of

iterations (see the example in the `example` folder). If something goes wrong for any of the images, explain what the problem might be.

Your code should contain a function `relaxed = relaximage(image, niters)` that takes as input a grayscale image matrix with pixel values lying in the range $[0, 1]$ (important!), and the number of iterations of relaxation to perform. It should return the relaxed image.