Elementary Facial Detection

Gregory Hughes, Mardigon Toler

Introduction

- Our proposed project was finding facial features
- Due to difficulty of the problem, goals changed
    - We now try to detect whether a face at all in an image
        - Same as our original basic goal
    - If the image in question contains a forward-facing face with sufficient lighting, we can easily find where the eyes are
- Also, in our pursuit, we have developed a useful tool
    - Detecting the thickness, or “strength” of edges

Implementation - Python

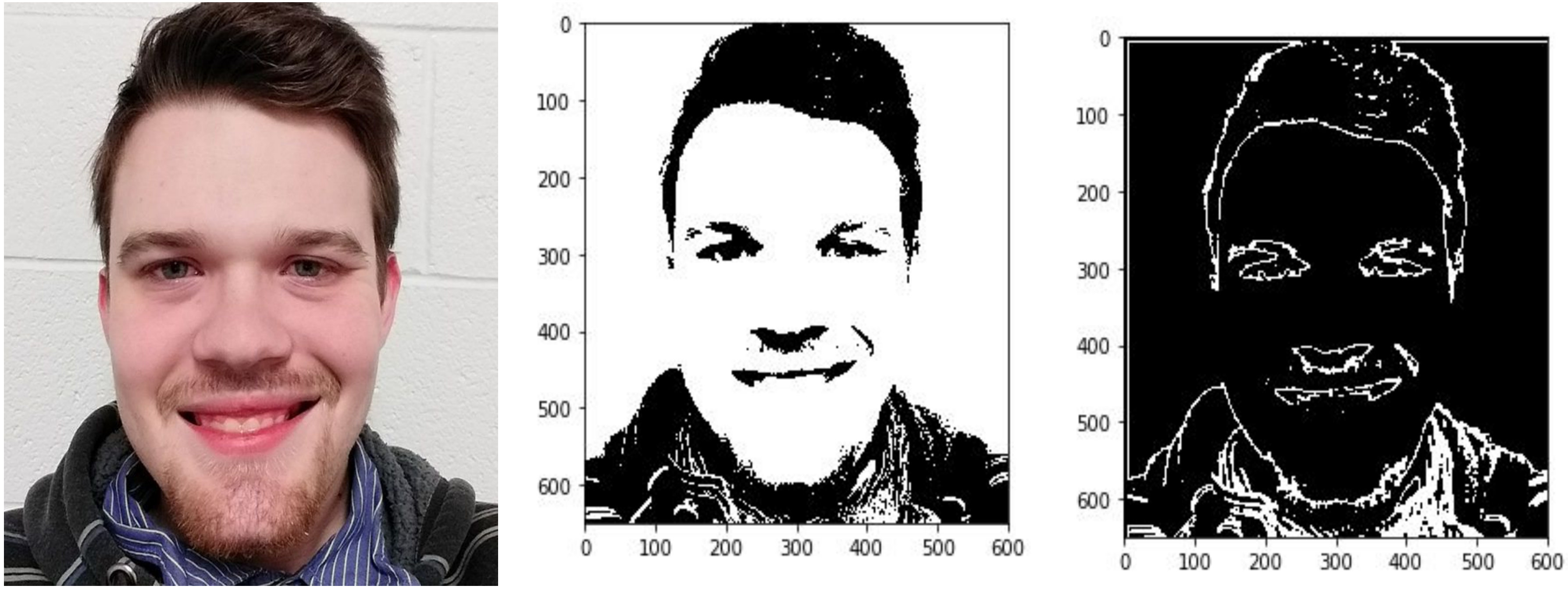
- The scikit-image library was used to open, load, and perform template matching on images
- The Numpy library was used because all images in scikit-image are Numpy-style arrays rather than the default Python arrays
- The functions for erosion and dilation were adapted from the 4th assignment.
- Most of the other concepts used (hit or miss/closing) were implemented with our erosion and dilation functions.

- Techniques inspired by *Image features extraction using mathematical morphology*
    - Marcin Iwanowski, Sławomir Skoneczny, Jarosław Szostakowski

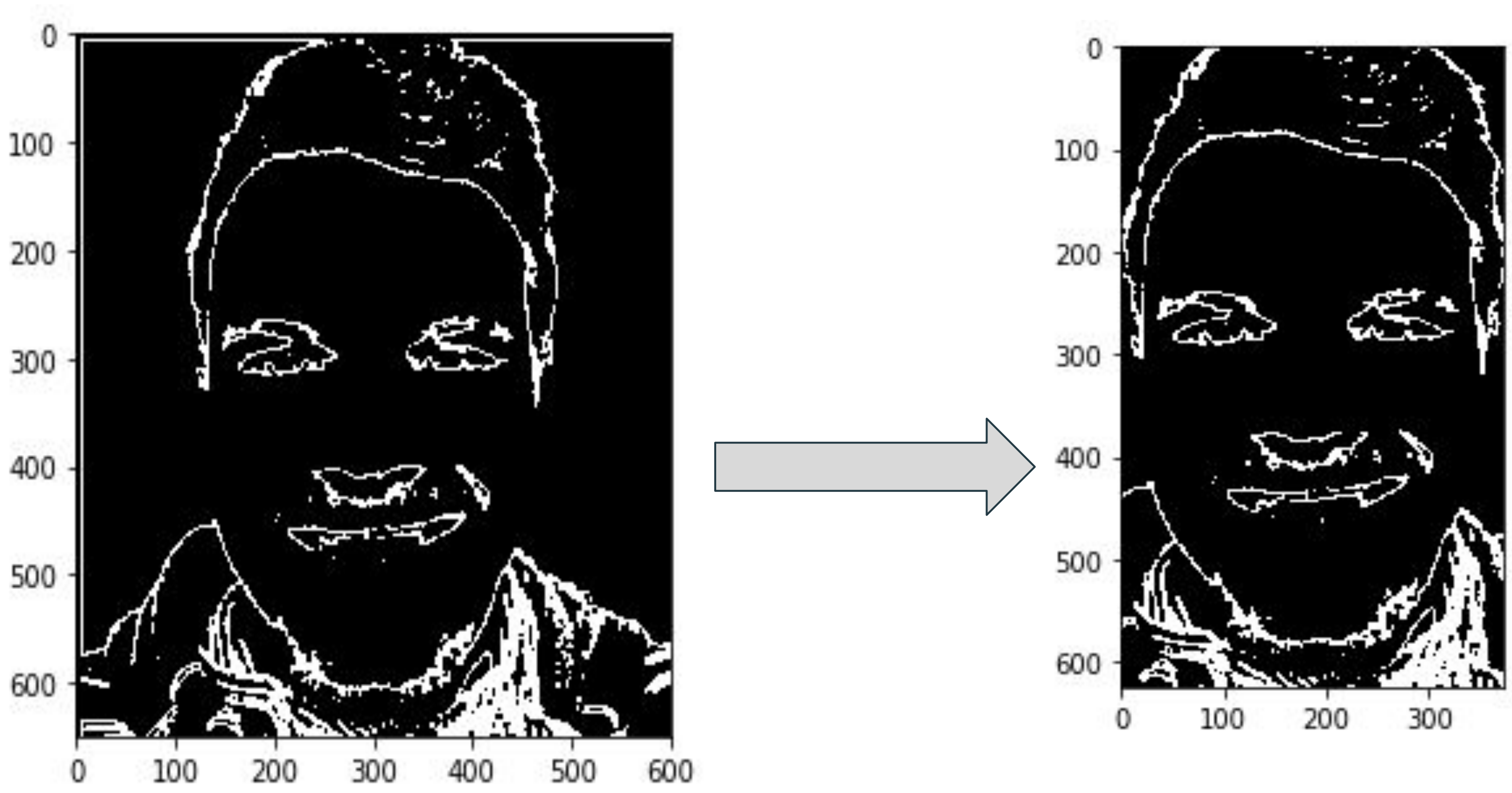
- We apply morphological processing to our binary images to find a region-interest to search for eyes.
- We then use template matching (provided by scikit-image) to find where the eyes are.

Process (Good Example):

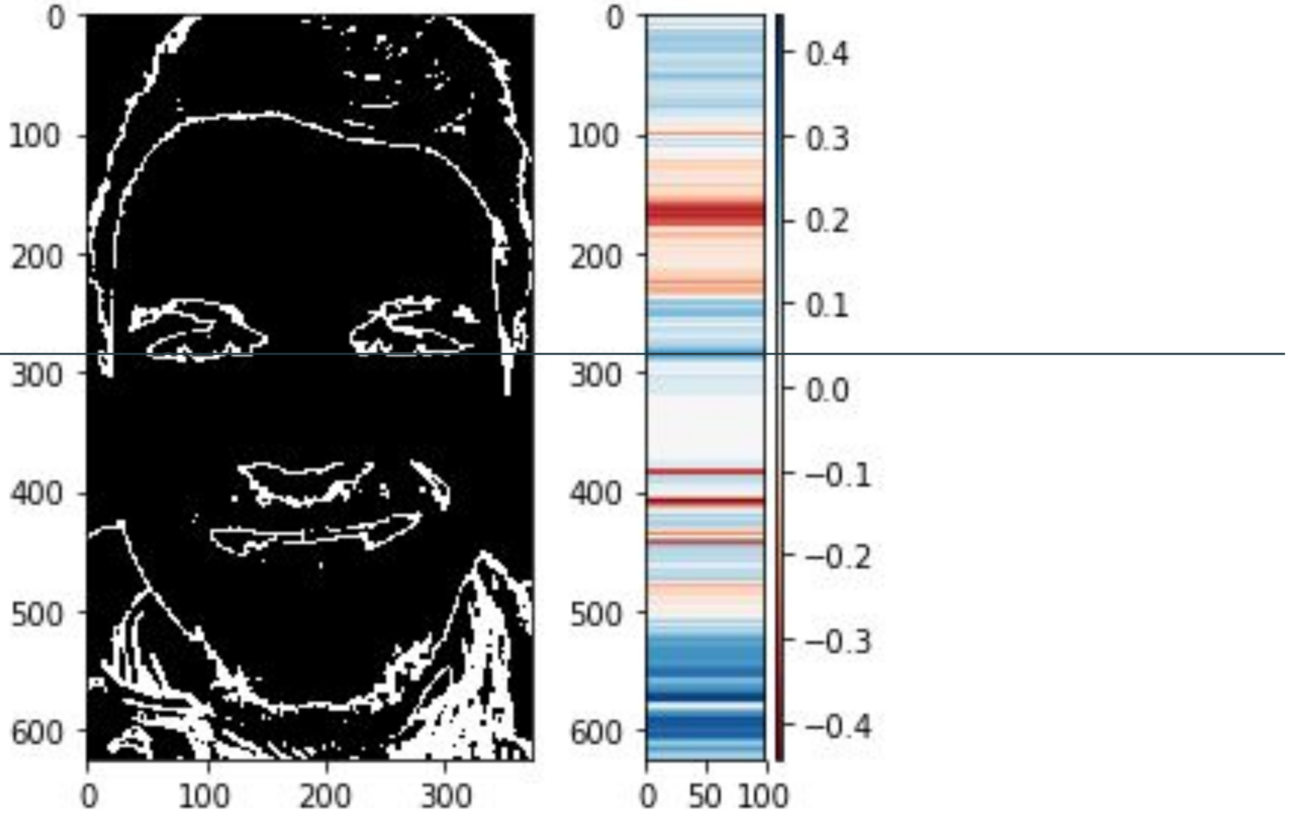
- We implemented all morphological processing used
  - Some were even modified for use in our Edge Strength detection, shown later
- From an image, we prepare it for a search for eyes



(Left to right): Original image, binary thresholding, morphologically closed hit-or-miss transform

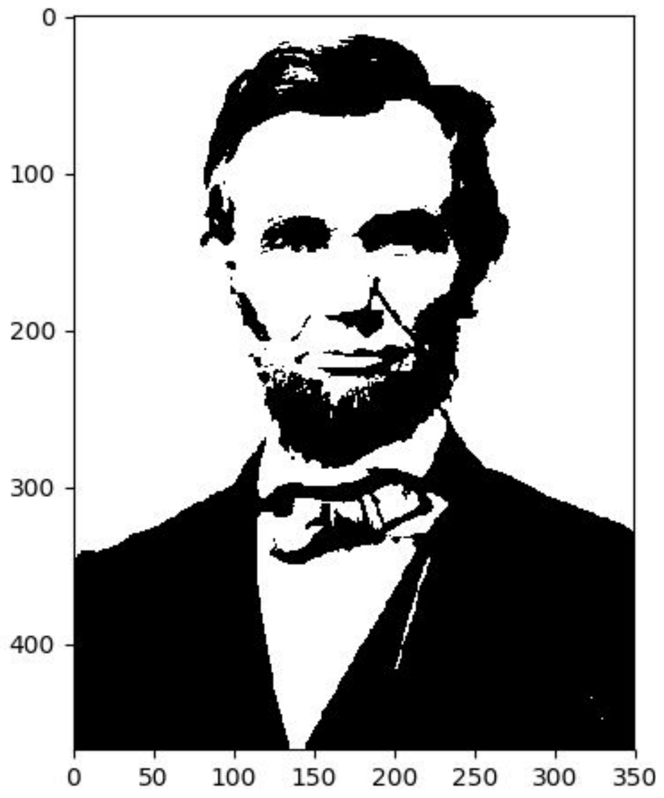


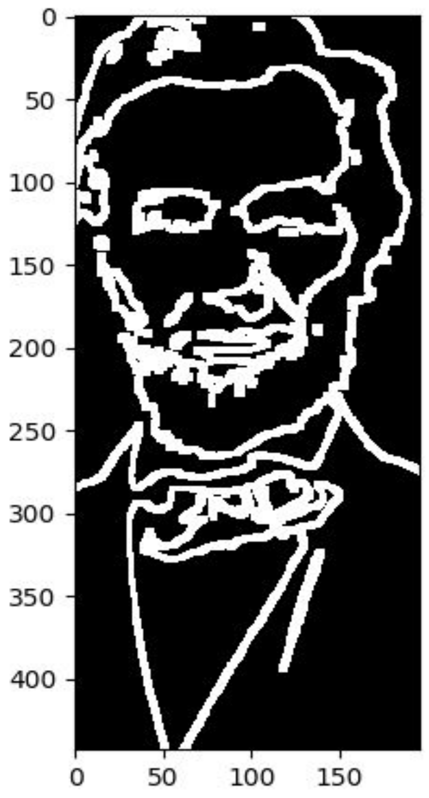
Finding Region of Interest

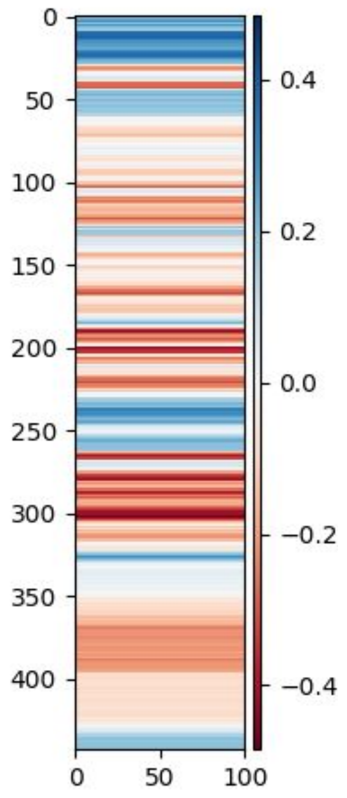


Likelihood of each row matching a template for eyes,
Give correct position for eyes if we ignore bottom of image

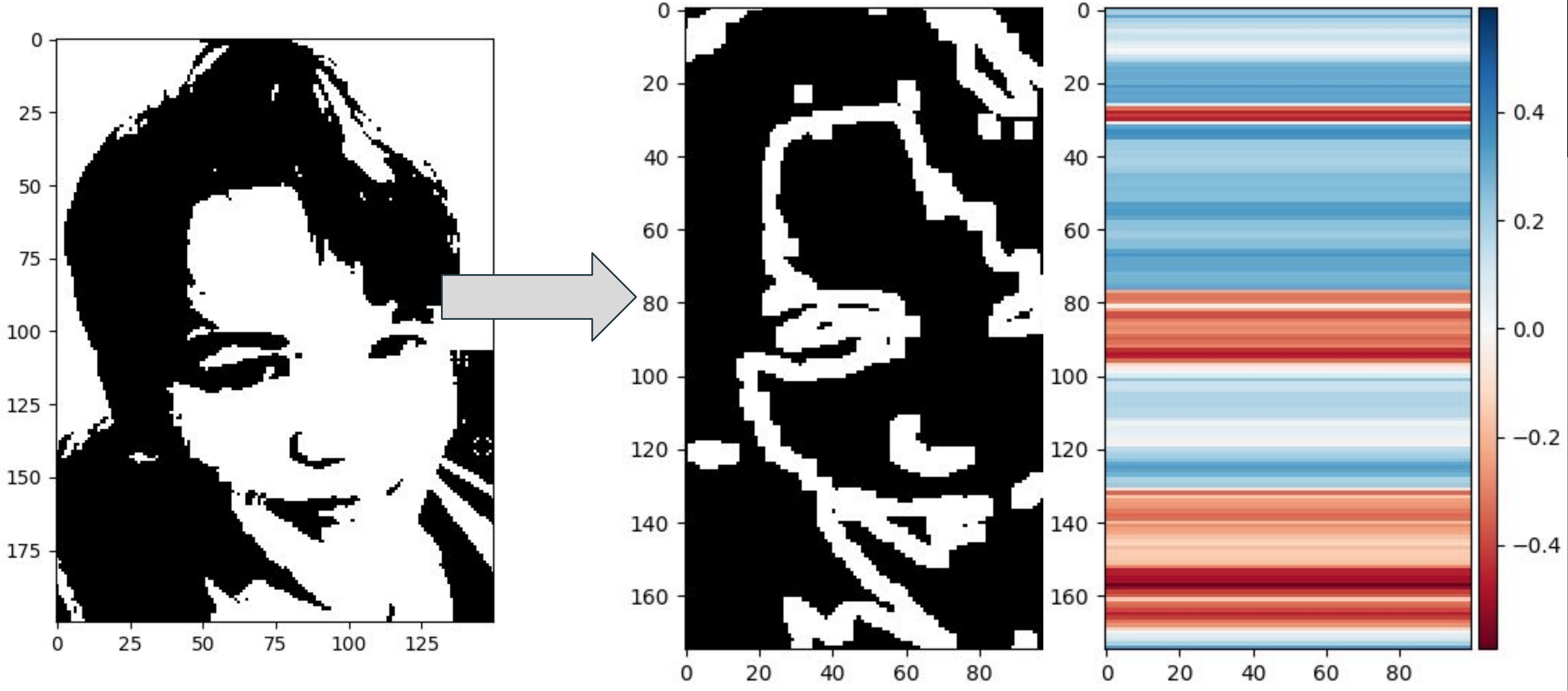
Example, not as good:







Bad Example


0
25
50
75
100
125
150
175
0
20
40
60
80
100
120
140
160
0.4
0.2
0.0
−0.2
−0.4

Sources

- *Image features extraction using mathematical morphology*
    - Marcin Iwanowski, Sławomir Skoneczny, Jarosław Szostakowski
- *Face Detection: A Survey*
    - Erik Hjelmås, Boon Kee Low
- *A novel method for automatic face segmentation, facial feature extraction and tracking*
    - Karin Sobotka, Ioannis Pitas

More Sources

- Python
- Scip-Py
  - Jones E, Oliphant E, Peterson P, *et al.* **SciPy: Open Source Scientific Tools for Python**, 2001-, http://www.scipy.org/
- Scikit-image
  - Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu and the scikit-image contributors. **scikit-image: Image processing in Python**
- Numpy
  - Travis E, Oliphant. **A guide to NumPy**, USA: Trelgol Publishing, (2006).

“Edge Strength Detection”

Edge Strength Detection

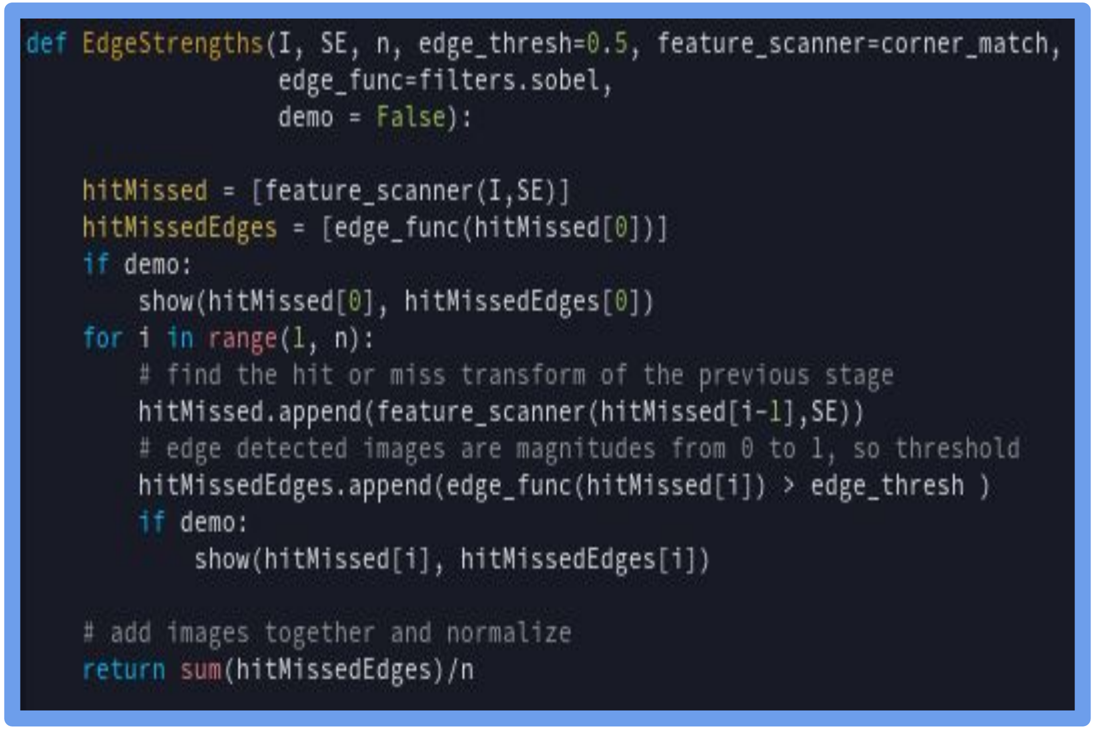
- From the papers we read we saw an opportunity for a new way to find edges using existing edge detection in an interesting way
- Our function creates an image of the edges in the input image. **The intensity of the edges is proportional to the thickness of the object with that edge**
- However, we have not used this in our facial detection yet

Different Hit-Or-Miss

- First, we modify the hit or miss transform
- It will mark corners rather than the center
    - (in these demos, it marks the upper left corner, but this can be reconfigured)
    - We will use a solid 3x3 foreground structuring element but this can also be changed

Process

- Apply the modified hit or miss transform to an image multiple times
- Find edges at edges at each step and save them
- Finally, sum the edge images and normalize



```
def EdgeStrengths(I, SE, n, edge_thresh=0.5, feature_scanner=corner_match,
                  edge_func=filters.sobel,
                  demo = False):

    hitMissed = [feature_scanner(I,SE)]
    hitMissedEdges = [edge_func(hitMissed[0])]
    if demo:
        show(hitMissed[0], hitMissedEdges[0])
    for i in range(1, n):
        # find the hit or miss transform of the previous stage
        hitMissed.append(feature_scanner(hitMissed[i-1],SE))
        # edge detected images are magnitudes from 0 to 1, so threshold
        hitMissedEdges.append(edge_func(hitMissed[i]) > edge_thresh )
        if demo:
            show(hitMissed[i], hitMissedEdges[i])

    # add images together and normalize
    return sum(hitMissedEdges)/n
```

Demonstration

Questions?