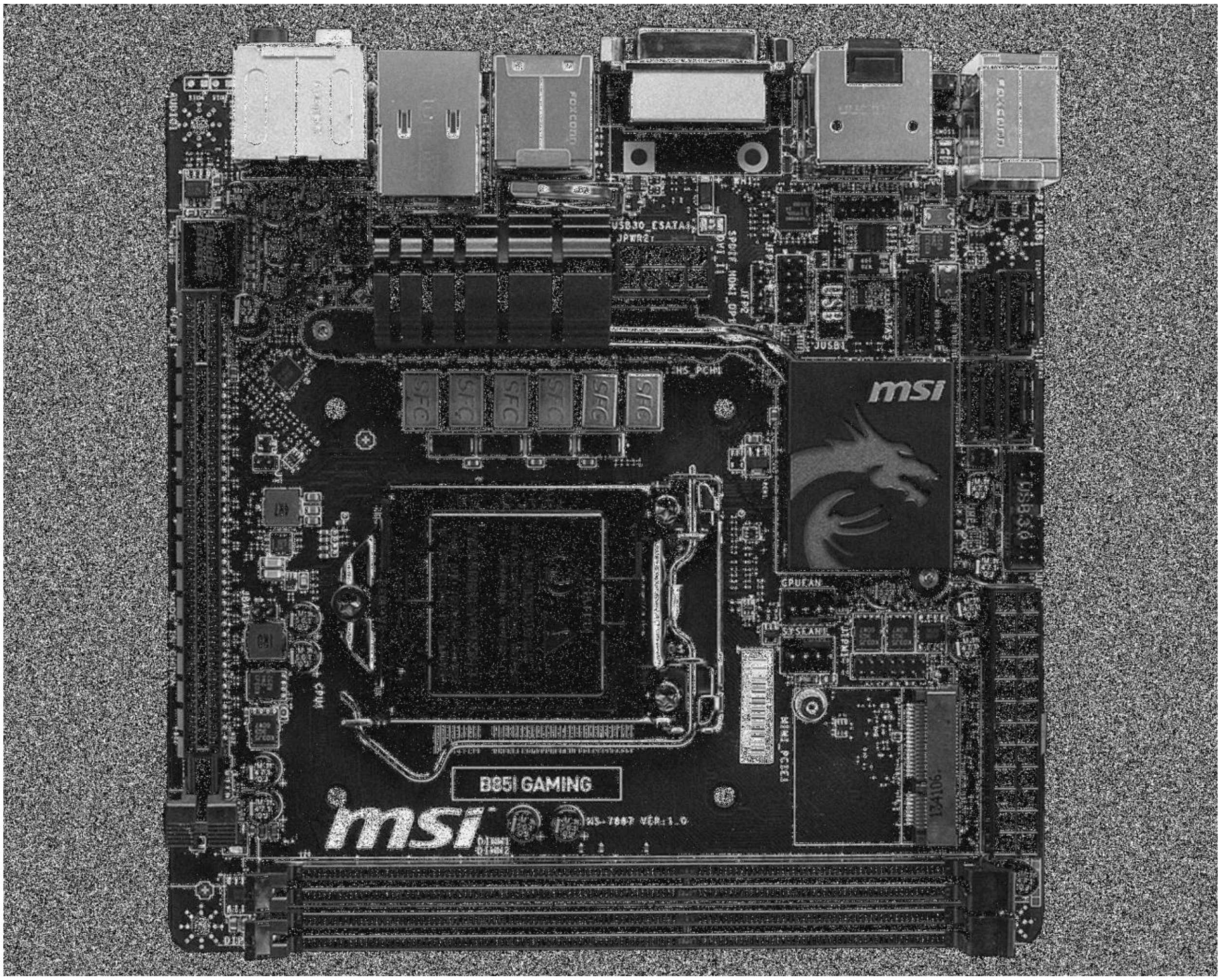
Richard Ngo

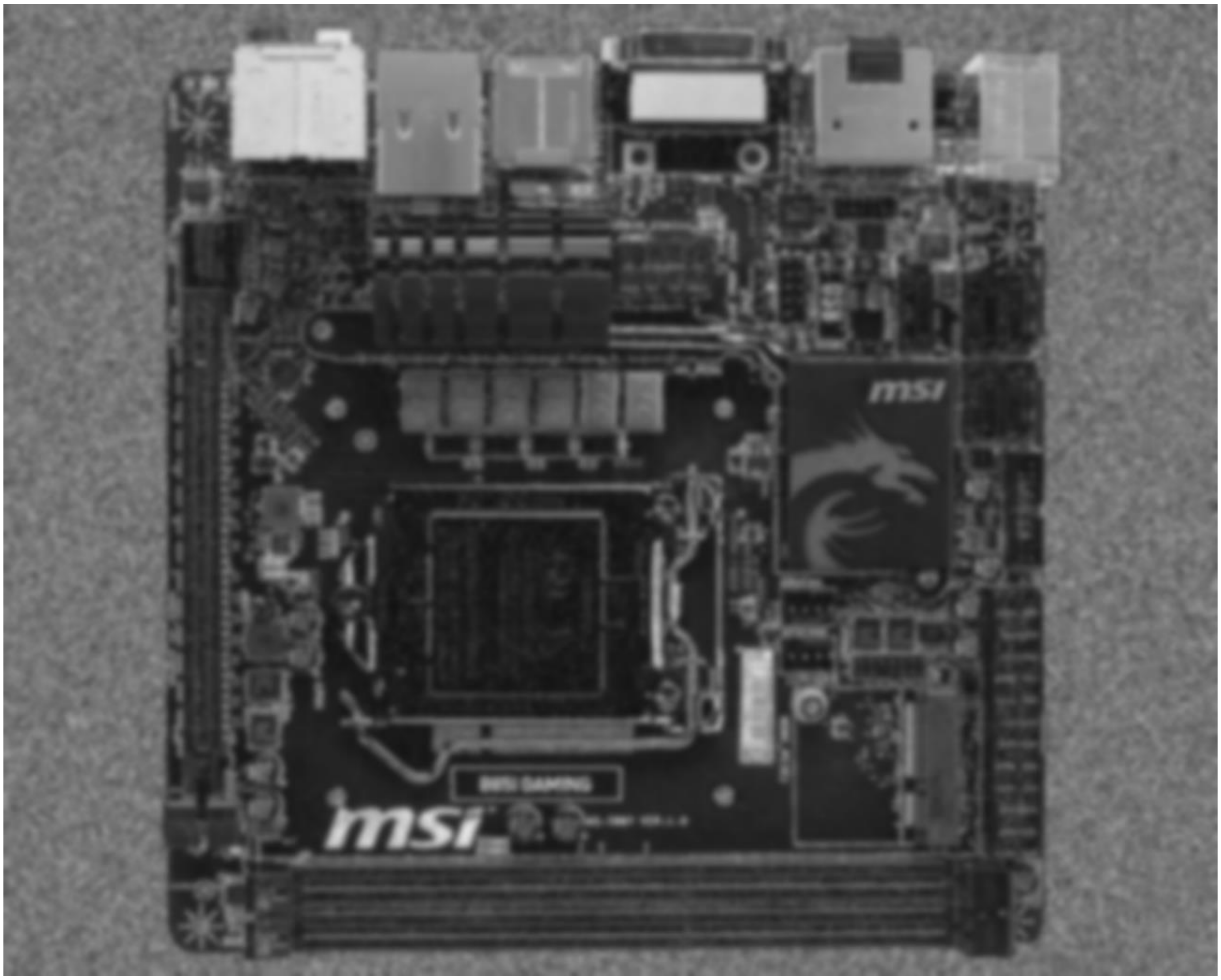
Part 1

Noisy Image (Noise Level = 10)



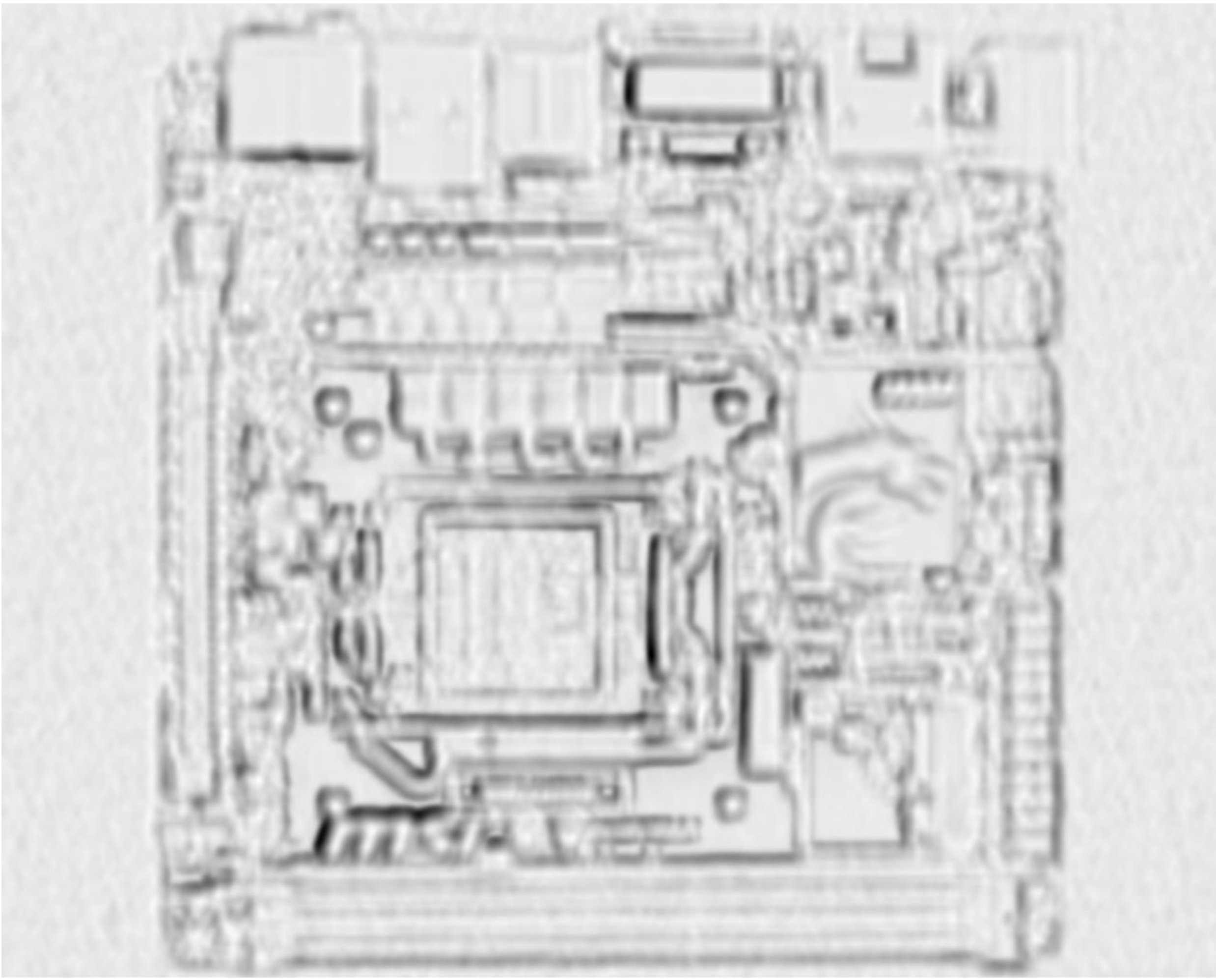
Richard Ngo

Smoothed Image (Sigma = 5)



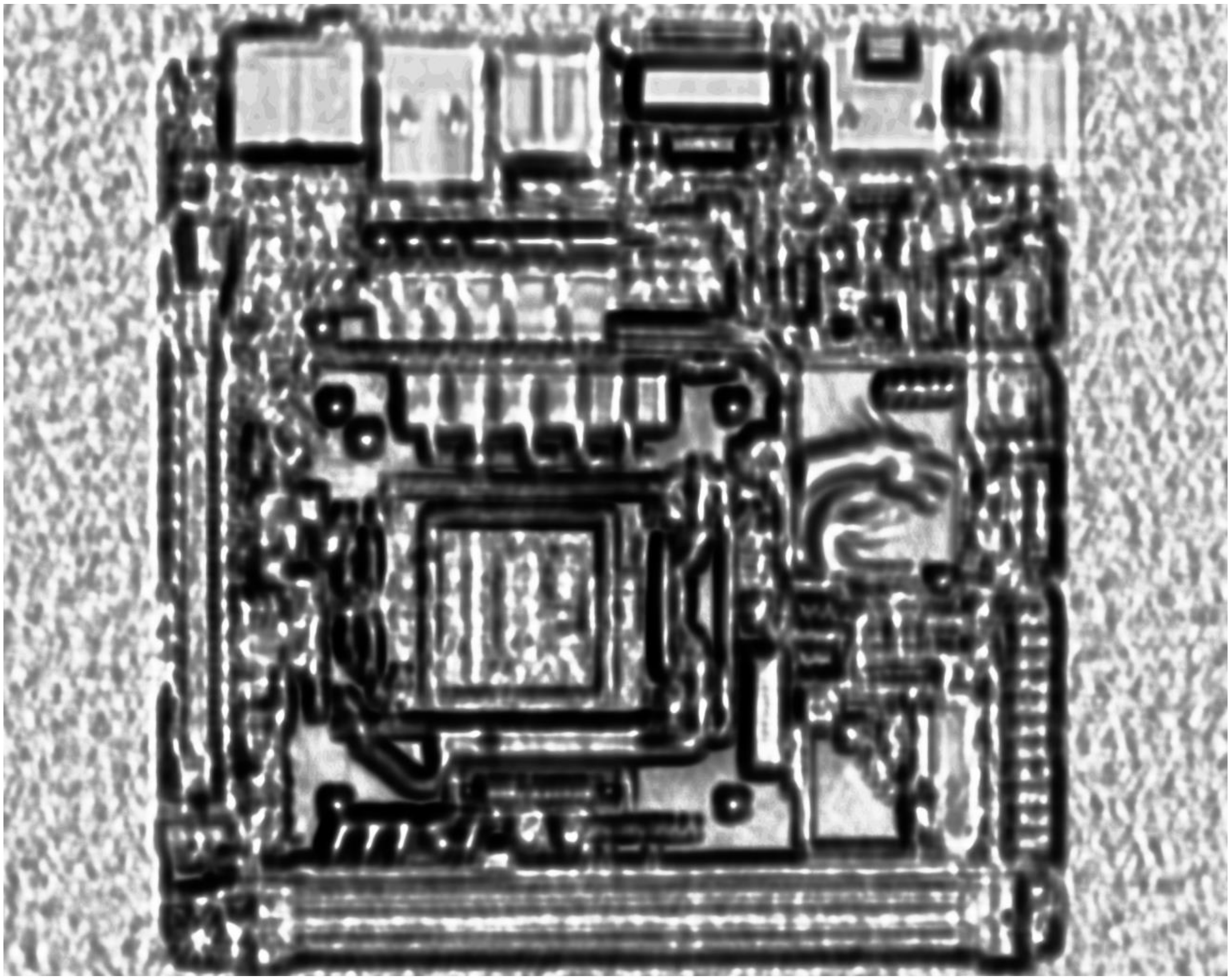
matchTemplate() Output

Normalized between Intensities of 0 and 255

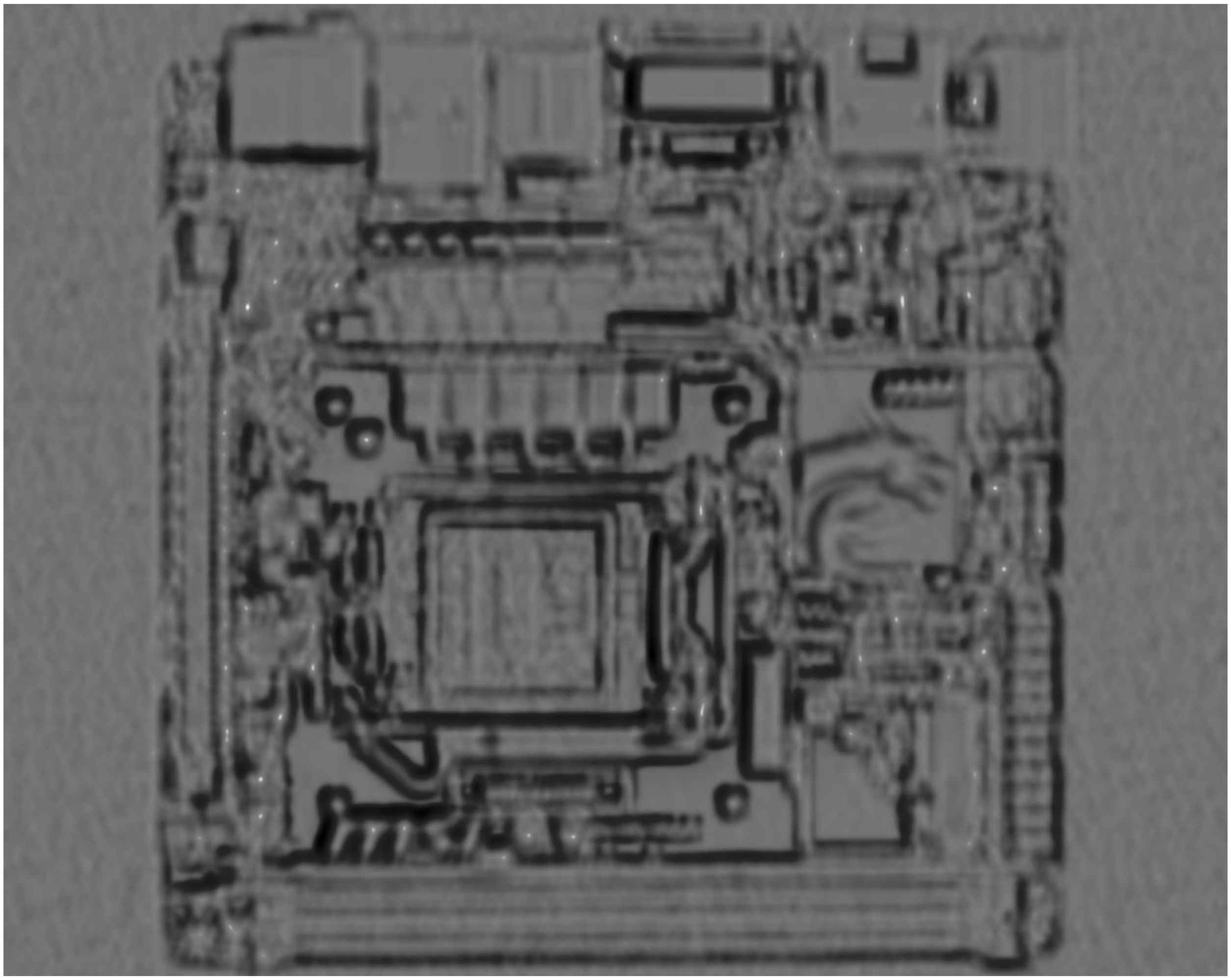


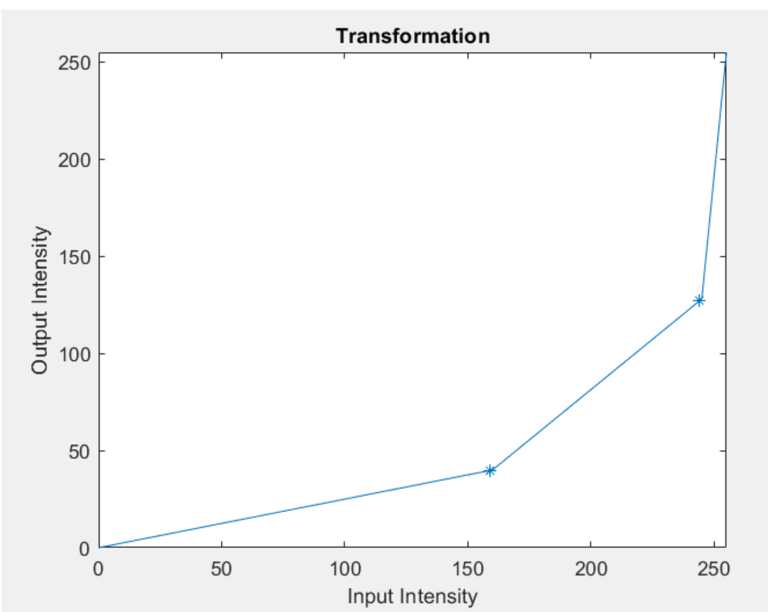
Richard Ngo

Histogram Equalized



Piecewise-Linear Transformed





Richard Ngo

Correlation

Noise \ Sigma	0	1	2	3	4	5
0	0.965792	0.969891	0.917495	0.873248	0.852239	0.839839
1	0.913549	0.955916	0.912018	0.871283	0.851057	0.839282
2	0.878104	0.945819	0.898524	0.868345	0.851431	0.839839
3	0.846237	0.937143	0.893791	0.862885	0.850053	0.840805
4	0.835504	0.93127	0.893636	0.863047	0.849373	0.839351
5	0.83463	0.923538	0.891087	0.862224	0.847441	0.840271
6	0.834473	0.916301	0.888987	0.860618	0.847275	0.837298
7	0.832688	0.917328	0.879976	0.860357	0.846619	0.837161
8	0.831505	0.911404	0.882923	0.859721	0.84524	0.838907
9	0.8338	0.902964	0.877366	0.85718	0.846113	0.836891
10	0.831968	0.911185	0.876956	0.857602	0.844551	0.835086

X Distance from Correct Location

Noise \ Sigma	0	1	2	3	4	5
0	0	0	0	0	46	45
1	62	62	0	0	46	45
2	2	62	0	0	46	45
3	62	0	2	46	45	45
4	174	62	62	24	45	45
5	174	62	62	1157	24	25
6	174	62	2	45	45	45
7	174	62	62	1157	46	45
8	391	62	62	1157	1157	45
9	174	62	62	1157	45	45
10	888	62	24	24	1157	45

Y Distance from Correct Location

Noise \ Sigma	0	1	2	3	4	5
0	0	0	0	0	105	106
1	82	82	0	0	105	106
2	67	82	0	0	106	106
3	82	0	66	105	105	106
4	177	82	82	924	105	106
5	176	82	82	396	864	864
6	177	82	67	105	105	105
7	177	82	82	396	105	105
8	228	82	82	396	396	106
9	176	82	82	396	105	106
10	476	82	864	864	396	106

At low levels of noise, adding a smoothing of sigma 2 or greater seems to decrease the maximum possible correlation between the template and the image. This is due to the fact that, assuming that the template itself comes directly from the image and contains little noise, an image with only few alterations to it still has the potential to fit the template almost perfectly, and altering the image at this point may differentiate it from template even further and decrease correlation. However, higher levels of noise have a chance to obscure the image and distract the template from matching up correctly in the image. This is why with the addition of noise, the max correlation from a 0 sigma smoothing (no smoothing at all) starts to drop off significantly with every additional level of noise. In this case, smoothing the image, and changing each pixel to be a select accumulation of its surrounding pixels may decrease the effect of each individual bit of noise overall. This effect can be seen in the significant increase in correlation when a smoothing of sigma 1 or more is used with an image with added noise. From the results, it seems that a smoothing of sigma 1 is the optimal value to use in terms of maximizing correlation for the examined data. However, it can also be noted that a high correlation does not always indicate the optimal sigma for smoothing in regard to matching the template to the correct location. When viewing the error in location caused by each combination, sigma values of 2 and 3 will result in more correct matches for noise levels of 1 and 2, than a sigma of 1 would, despite having lower correlations. This may be possible due to the fact that, though noise can lower correlation, it also has the capacity contain a similar enough orientation to the template, and trick a program into calculating a larger correlation than is deserved. Overall, gaussian filters can be used to limit the high changes in intensity that usually comes with noise, while mostly preserving low changes of intensities in the original image. This comes at the cost of significantly altering the high changes in intensity in the original image as well, and can have the potential to hurt the effectiveness of an template matching program if not used in moderation.

Code Used

```
# CompVisHw2_1.py - Compares effectiveness of template matching with different levels of blur and noise

# Created on 10/20/19

# @author: Richard Ngo

import numpy as np, cv2


################################

#noisy - modified from Shubham Pachori on stackoverflow

def noisy(image, noise_type, sigma):

  if noise_type == "gauss":

    row,col = image.shape

    mean = 0
```

```
        gauss = np.random.normal(mean,sigma,(row,col))
        gauss = gauss.reshape(row,col)
        noisy = image + gauss
        return noisy
    elif noise_type == "s&p":
        row,col = image.shape
        s_vs_p = 0.5
        amount = 0.004
        out = np.copy(image)
        # Salt mode
        num_salt = np.ceil(amount * image.size * s_vs_p)
        coords = [np.random.randint(0, i - 1, int(num_salt))
            for i in image.shape]
        out[coords] = 1
        # Pepper mode
        num_pepper = np.ceil(amount* image.size * (1. - s_vs_p))
        coords = [np.random.randint(0, i - 1, int(num_pepper))
            for i in image.shape]
        out[coords] = 0
        return out
    elif noise_type == "poisson":
        vals = len(np.unique(image))
        vals = 2 ** np.ceil(np.log2(vals))
        noisy = np.random.poisson(image * vals) / float(vals)
        return noisy
    elif noise_type =="speckle":
        row,col = image.shape
        gauss = np.random.randn(row,col)
        gauss = gauss.reshape(row,col)
```

```
        noisy = image + image * gauss
    return noisy
################################

img = cv2.imread('motherboard-gray.png', cv2.IMREAD_GRAYSCALE)
temp = cv2.imread('template.png', cv2.IMREAD_GRAYSCALE)

heightI = len(img)
widthI = len(img[0])

heightT = len(temp)
widthT = len(temp[0])

maxnoise = 10
maxsig = 5
maxcorr = np.zeros((maxnoise+1, maxsig+1), dtype = "float32")
maxlocy = np.zeros((maxnoise+1, maxsig+1), dtype = "int64")
maxlocx = np.zeros((maxnoise+1, maxsig+1), dtype = "int64")
for N in range(maxnoise+1):
    for S in range(maxsig+1):
        no = np.uint8(noisy(img, 'gauss', N))
        if S == 0:
            nosm = no
        else:
            nosm = cv2.GaussianBlur(no,(S*6+1,S*6+1), S)
        matched = cv2.matchTemplate(nosm,temp,cv2.TM_CCORR_NORMED)
        max_val = np.amax(matched)
        loc = np.where(matched==max_val)
```

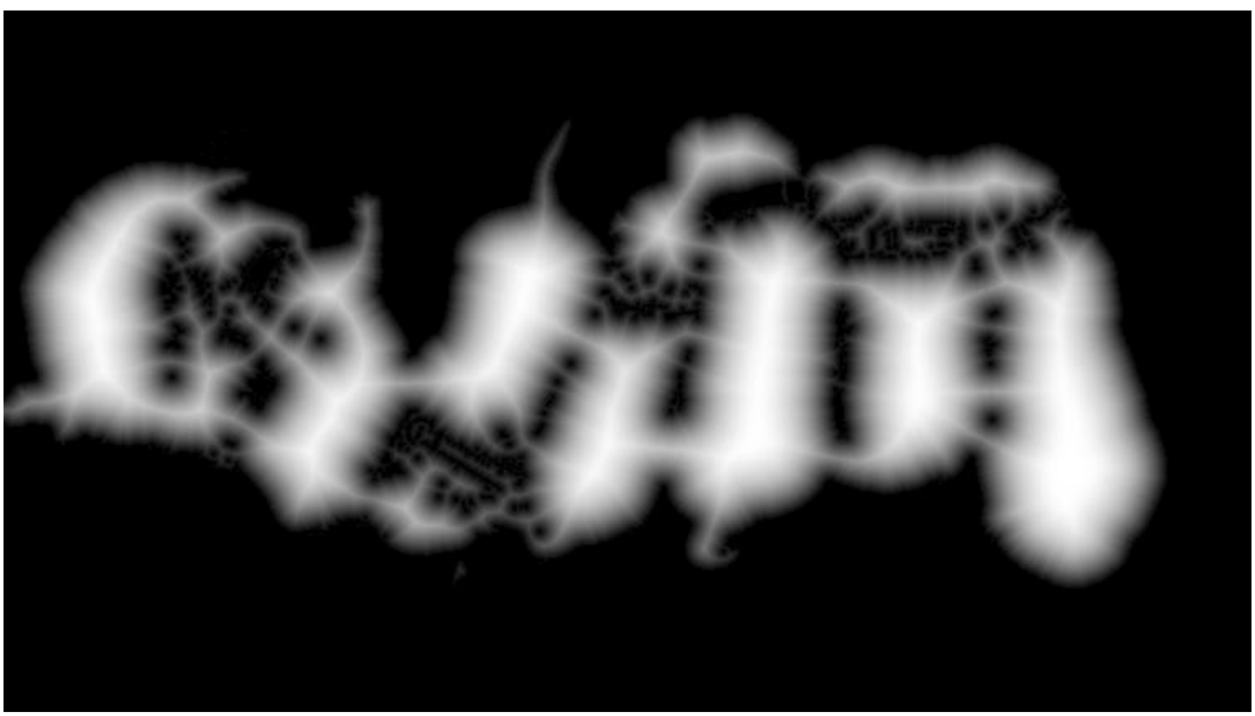
```
        #for viewing
        maxcorr[N,S] = max_val
        maxlocy[N,S] = abs(loc[0][0]-382)
        maxlocx[N,S] = abs(loc[1][0]-438)

N = maxnoise
S = maxsig

heightM = len(matched)
widthM = len(matched[0])
dst = np.zeros(shape=(len(matched),len(matched[0])))
matched = np.uint8(cv2.normalize(matched,dst,0,255,cv2.NORM_MINMAX))
histEqmatched = cv2.equalizeHist(matched)
transmatch = np.zeros((heightM, widthM), dtype = "uint8")

#Piecewise-Linear Transformation
r1 = 160#input
s1 = 40#output
r2 = 245#input
s2 = 128#output

for i in range(heightM):
    for j in range(widthM):
        if matched[i,j] <= r1:
            transmatch[i,j] = s1/r1*matched[i,j]#if intensity of image is below or equal to r1
        elif matched[i,j] >= r2:
            transmatch[i,j] = (255-s2)/(255-r2)*(matched[i,j]-r2)+s2#if intensity of image is above or equal to
r2
        else:
```

```
        transmatch[i,j] = (s2-s1)/(r2-r1)*(matched[i,j]-r1)+s1#if intensity of image is between r1 and r2

heightI = len(nosm)
widthI = len(nosm[0])

#show resulting images for N = 10, S = 5
cv2.namedWindow('Noise: N='+str(N)+' S='+str(S), flags=cv2.WINDOW_NORMAL)
cv2.imshow('Noise: N='+str(N)+' S='+str(S), no)
cv2.resizeWindow('Noise: N='+str(N)+' S='+str(S), (int(widthI/2), int(heightI/2)))

cv2.namedWindow('Smoothed: N='+str(N)+' S='+str(S), flags=cv2.WINDOW_NORMAL)
cv2.imshow('Smoothed: N='+str(N)+' S='+str(S), nosm)
cv2.resizeWindow('Smoothed: N='+str(N)+' S='+str(S), (int(widthI/2), int(heightI/2)))

cv2.namedWindow('Corr: N='+str(N)+' S='+str(S), flags=cv2.WINDOW_NORMAL)
cv2.imshow('Corr: N='+str(N)+' S='+str(S), matched)
cv2.resizeWindow('Corr: N='+str(N)+' S='+str(S), (int(widthM/2), int(heightM/2)))

cv2.namedWindow('HECorr: N='+str(N)+' S='+str(S), flags=cv2.WINDOW_NORMAL)
cv2.imshow('HECorr: N='+str(N)+' S='+str(S), histEqmatched)
cv2.resizeWindow('HECorr: N='+str(N)+' S='+str(S), (int(widthM/2), int(heightM/2)))

cv2.namedWindow('LTCorr: N='+str(N)+' S='+str(S), flags=cv2.WINDOW_NORMAL)
cv2.imshow('LTCorr: N='+str(N)+' S='+str(S), transmatch)
cv2.resizeWindow('LTCorr: N='+str(N)+' S='+str(S), (int(widthM/2), int(heightM/2)))



cv2.waitKey(0)
cv2.destroyAllWindows()
```

Part 2

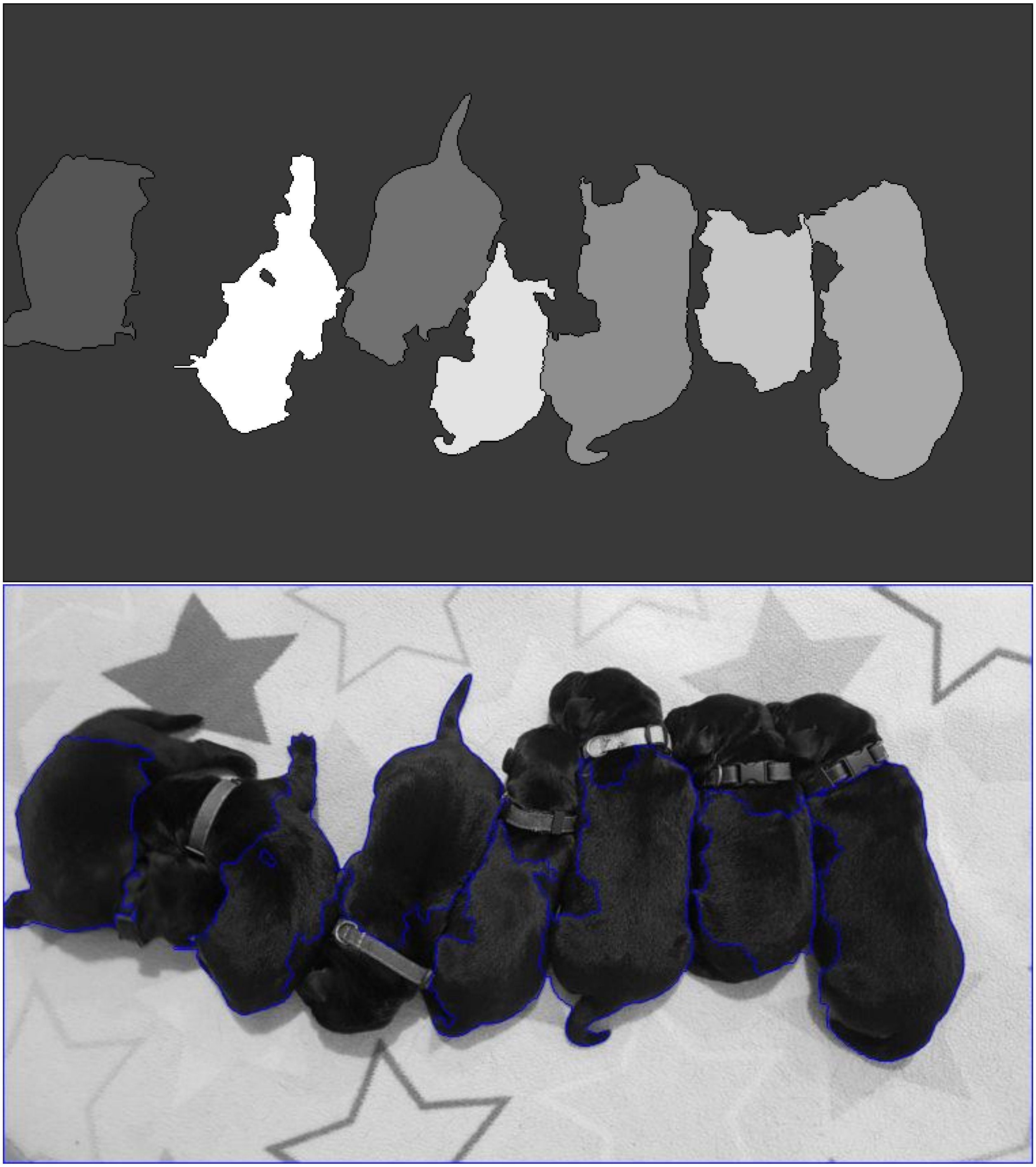
Original Image

Distance Image



Richard Ngo

Segmentated Image



Code Used

CompVisHw2_2.py - Setting up puppies for watershed seperation

```
# Created on 10/21/19

# @author: Richard Ngo


import cv2

import numpy as np


img = cv2.imread('puppies.png',0)


#set bright boundaries to backround

ret,backround = cv2.threshold(img,105,255,cv2.THRESH_TOZERO_INV)#70, 105

#set dark boundaries between dogs to backround

ret,backround = cv2.threshold(backround,9,255,cv2.THRESH_BINARY)#9, 10


#cv2.imshow('backround', backround)


distIm = cv2.distanceTransform(backround,cv2.DIST_L2,5)


dst = np.zeros(shape=(len(distIm),len(distIm[0])))


#cv2.imshow('distanceT', np.uint8(cv2.normalize(distIm,dst,0,255,cv2.NORM_MINMAX)))


cv2.imshow('distance transform',
cv2.equalizeHist(np.uint8(cv2.normalize(distIm,dst,0,255,cv2.NORM_MINMAX))))


ret, foreground = cv2.threshold(distIm,0.55*distIm.max(),255,0)

#cv2.imshow('foreground', foreground)


foreground = np.uint8(foreground)

backround = np.uint8(backround)
```

```
#code based on opencv watershed tutorial from here on out
#https://docs.opencv.org/master/d3/db4/tutorial_py_watershed.html

#to be flooded
unknown = cv2.subtract(backround,foreground)

#cv2.imshow('unknown', unknown)

#markers
ret, markers = cv2.connectedComponents(foreground)

# make backround the shallowest segment
markers = markers+1

# set up region to be flooded
markers[unknown==255] = 0

#cv2.imshow('markers', np.uint8(np.uint8(cv2.normalize(markers,dst,0,255,cv2.NORM_MINMAX))))

img = cv2.imread('puppies.png')
cv2.imshow('Original Image', img)
#blur image for more effective segmentation
imgGB = cv2.GaussianBlur(img,(5,5),5)#9,9,1  5,5,5  11,11,3  11,11,1

markers = cv2.watershed(imgGB,markers)

"""
```

```
#layered watershed?
test = markers
unknownT = unknown

unknownT[test > 1] = 0
test[unknownT==255] = 0
test[test == -1] = 0

#test[unknown==255] = 0
#test[unknown<255] = 1
#unknownT = unknown
#test[markers == -1] = 0
#test[markers <= 1] = 0
#unknownT[test == 255] = 0
#test[unknownT==255] = 0

#ret, test = cv2.connectedComponents(test)

#test = cv2.watershed(imgGB,test)
#test = np.uint8(cv2.normalize(test,dst,0,255,cv2.NORM_MINMAX))

test = cv2.watershed(imgGB,test)



test = np.uint8(cv2.normalize(test,dst,0,255,cv2.NORM_MINMAX))
cv2.imshow('test', test)
"""

img[markers == -1] = [255,0,0]
```

```
cv2.imshow('Final Segmentations', np.uint8(cv2.normalize(markers,dst,0,255,cv2.NORM_MINMAX)))
cv2.imshow('Final Segmented Image', img)

print('number of segments found excluding backround',np.amax(markers)-1)

cv2.waitKey(0)
cv2.destroyAllWindows()
```