CS216 HW2 Name: Yifan Tian ID:78921267

1. Show that convolution is associative, that is that(f⋆g)⋆h=f⋆(g⋆h)

(f *(g*h))=((f *g)*h).
For convolution, we have:

$$f * g = \int_{-\infty}^{\infty} f(x)g(y-x)dx = \int_{-\infty}^{\infty} g(x)f(y-x)dx = g * f$$

According to **Fubini's theorem**:

$$\int_X \left(\int_Y f(x,y)\,\mathrm{d}y \right) \mathrm{d}x = \int_Y \left(\int_X f(x,y)\,\mathrm{d}x \right) \mathrm{d}y = \int_{X\times Y} f(x,y)\,\mathrm{d}(x,y)$$

Then we have:

$$\begin{aligned}
\big(f * (g * h)\big) &= \int_{-\infty}^{\infty} f(z-x)\left(\int_{-\infty}^{\infty} g(y)h(x-y)dy\right)dx \\
&= \int_{-\infty}^{\infty} f(z-x)\left(\int_{-\infty}^{\infty} g(x-y)h(y)dy\right)dx \\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(z-x)g(x-y)h(y)dy\,dx \\
&= \int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty} f(z-x)g(x-y)dx\right)h(y)dy \\
&= \int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty} f(z-y-\tau)g(\tau)d\tau\right)h(y)dy = \big((f * g) * h\big)
\end{aligned}$$

Some examples:

a = [1 2 3], b = [2 3 4], c = [3 4 5]

conv(conv(a,b),c) = 6 29 86 150 184 133 60
conv(a,conv(b,c)) = 6 29 86 150 184 133 60

2. Show that correlation is not associative

For correlation, we have:

$$f * g = \int_{-\infty}^{\infty} f(x)g(y+x)dx$$

But only for even f or g, we have:

$$f * g = \int_{-\infty}^{\infty} f(x)g(y+x)dx = \int_{-\infty}^{\infty} g(\tau)f(\tau-y)d\tau = \int_{-\infty}^{\infty} g(\tau)f(\tau+y)dx = g * f$$

So it is not generally true, then only when f and g are even or g and h are even then we have that:

$$\begin{aligned}
\big(f * (g * h)\big) &= \int_{-\infty}^{\infty} f(z+x)\left(\int_{-\infty}^{\infty} g(y)h(x+y)dy\right)dx \\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(z+x)g(y)h(x+y)dy\,dx \\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(z+x)g(x+y)h(y)dx\,dy \\
&= \int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty} f(z+x)g(x+y)dx\right)h(y)dy \\
&= \int_{-\infty}^{\infty}\left(\int_{-\infty}^{\infty} f(\tau)g(y-z+\tau)d\tau\right)h(y)dy = \big((f * g) * h\big)
\end{aligned}$$

A counter example:

a = [1 2 3], b = [2 3 4], c = [3 4 5]

Because to do correlation, we need to flip the vector, we use fliplr().

So we have:

Conv(conv(a,fliplr(b)),fliplr(c)) = 20 71 156 178 142 63 18

Conv(a, fliplr(conv(b,fliplr(c)))) = 12 49 124 174 170 89 30

3. If we have an image I of dimension HxW and we convolve it with a filter f of size MxN using the spatial domain formula given in class, what will the complexity be? How about if we use the FFT "trick"?

Because for each pixel, we need to convolve it with a filter of size N, so for each pixal, the computation cost is **M*N**, as each pixal within the window of M*N implement a multiplication. And the image I has dimension **H*W**. So the total complexity will be **H*W*M*N**.

If we use the FFT trick, then the computation of convolution will be **log(M*N)** instead of

M*N. So the total complexity will be **H*W*log(M*N).**

4. **If filter is the product of two functions, then we can compute the convolution more efficiently:**

$$f * g = \sum_{v=1}^{v=M} \sum_{u=1}^{u=N} f(u,v)g(x-u, y-v) \qquad (1)$$

$$= \sum_{v=1}^{v=M} \sum_{u=1}^{u=N} f_1(u)f_2(v)g(x-u, y-v)$$

$$= \sum_{v=1}^{v=M} f_2(v) \sum_{u=1}^{u=N} f_1(u)g(x-u, y-v) \qquad (2)$$

So equation (1) has higher computing cost than equation (2) as equation require M*N multiplications while equation (2) has M+N multiplications.

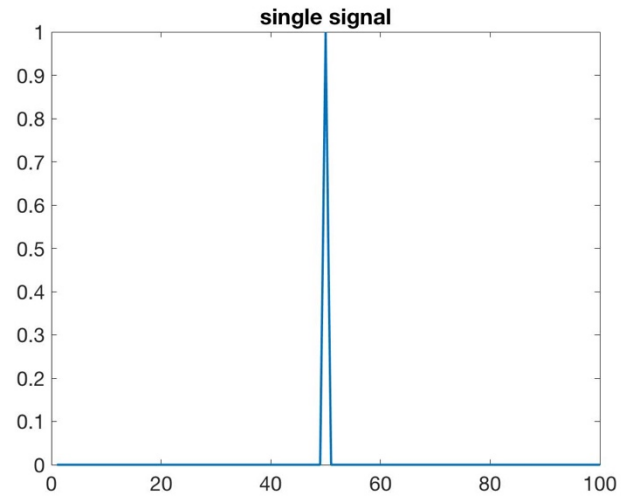
Becuause Gaussian is separable: the Gaussian of 2 variables is the multiplication of Gaussian of single variable: G(x,y) = G(x)*G(y), so following above method, we can decompose the convolution of Gaussian kernel and the total convolution will be the multiplication of 2 single convolutions. Then it is more efficient.

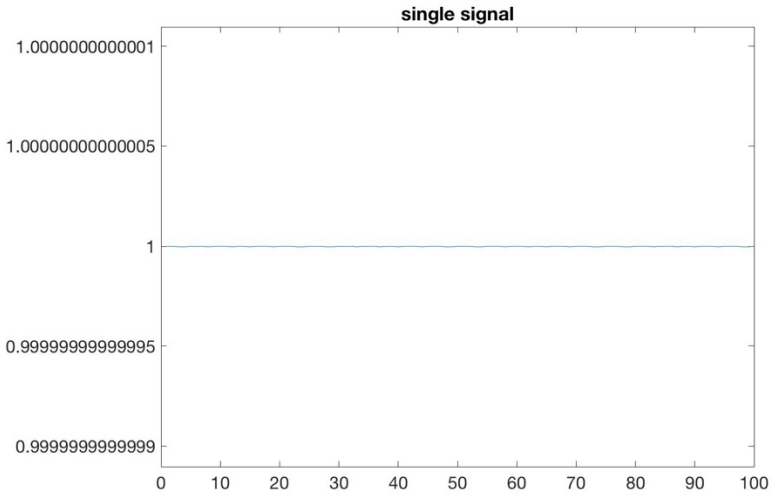
$$\begin{aligned} G_\sigma(x,y) &= \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \\ &= \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right) \cdot \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{y^2}{2\sigma^2}\right) \\ &= g_\sigma(x) \cdot g_\sigma(y) \end{aligned}$$

So when do convolution, we can decompose: $I * G = I * G(a) * G(b)$.

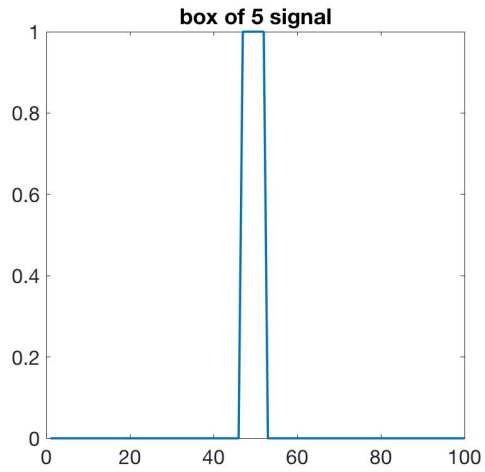
5. Experiments of DFT.

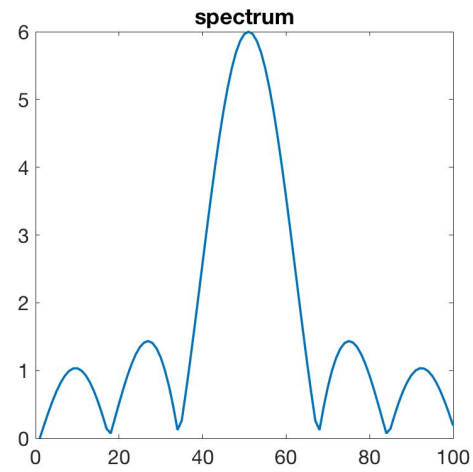
(1) Start out in 1D and make a signal which is of length 100 with a single impulse of height 1 in the middle.

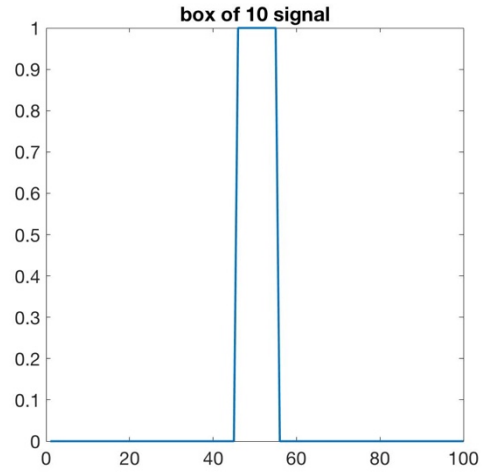


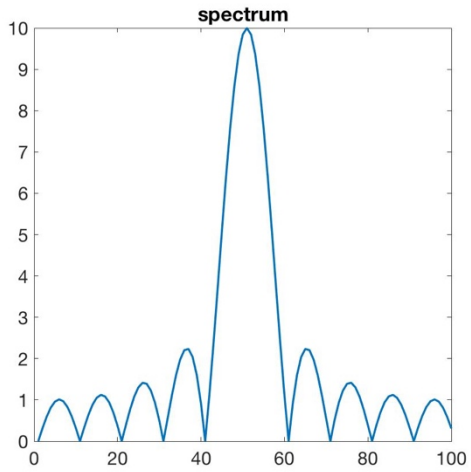


(2) Signal of unit height “box function” 5 and 10 samples long and corresponding DFT spectrum.









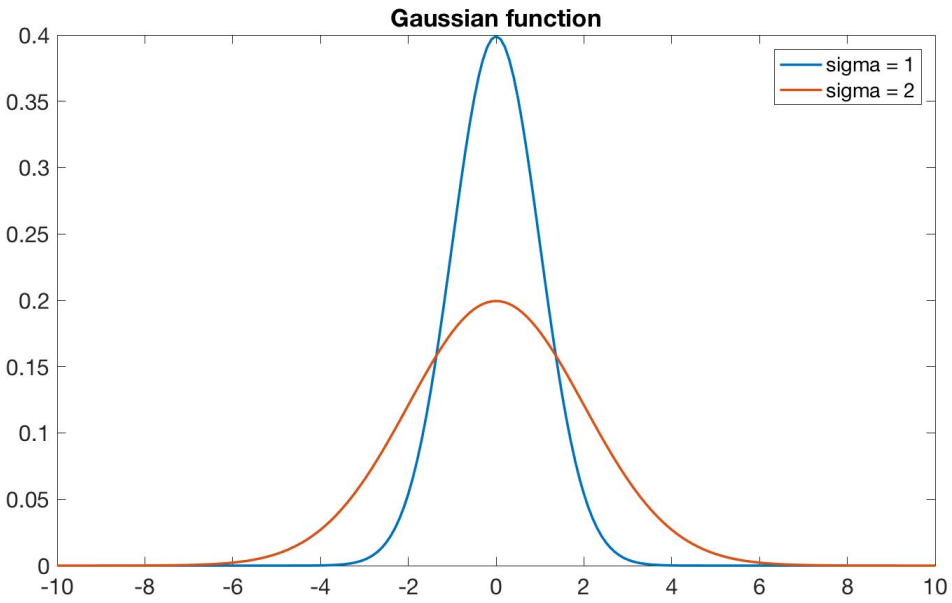
The number of signals is the number of peaks in frequency domain, also the number of signals is the height of the peak at frequency domain. All of these are consistent with the DFT equation:

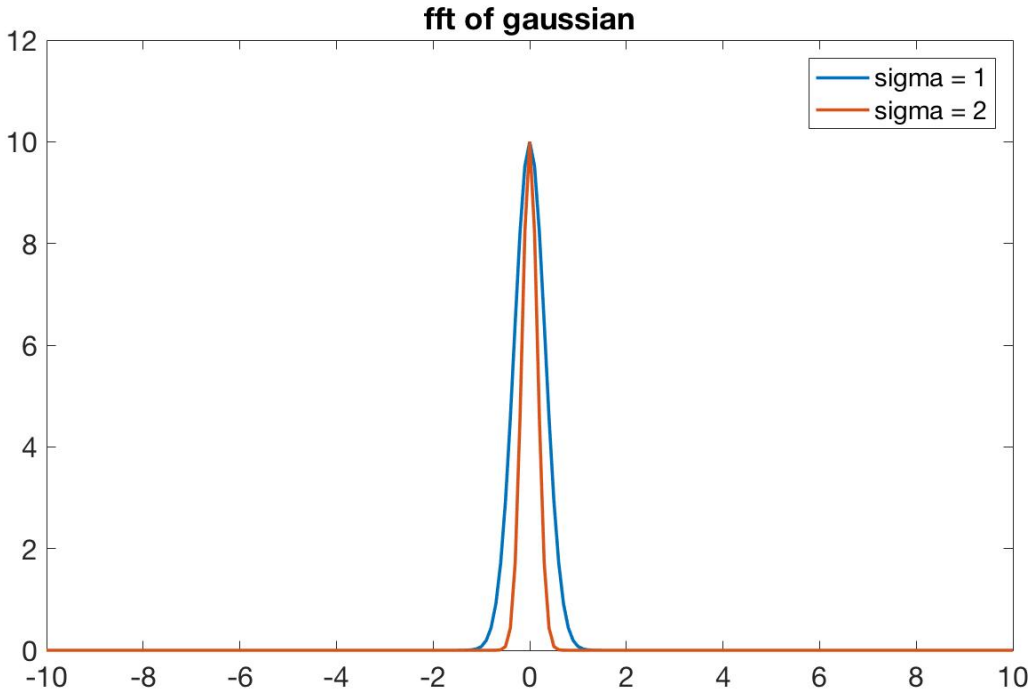
$$F(p,q) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(m,n)e^{-j2\pi pm/M}e^{-j2\pi qn/N} \quad \begin{matrix} p = 0,\ 1,\ \ldots,\ M-1 \\ q = 0,\ 1,\ \ldots,\ N-1 \end{matrix}$$

and

$$f(m,n) = \frac{1}{MN}\sum_{p=0}^{M-1}\sum_{q=0}^{N-1} F(p,q)e^{j2\pi pm/M}e^{j2\pi qn/N} \quad \begin{matrix} m = 0,\ 1,\ \ldots,\ M-1 \\ n = 0,\ 1,\ \ldots,\ N-1 \end{matrix}$$

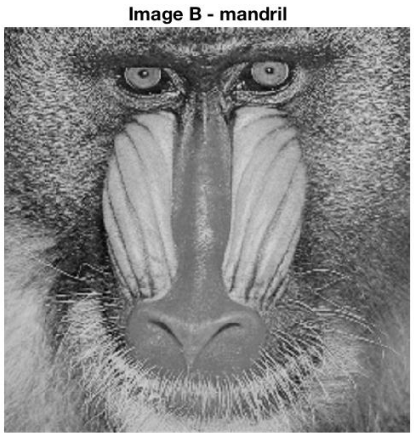
(3) Gaussian function $\sigma = 1$ and $\sigma = 2$.



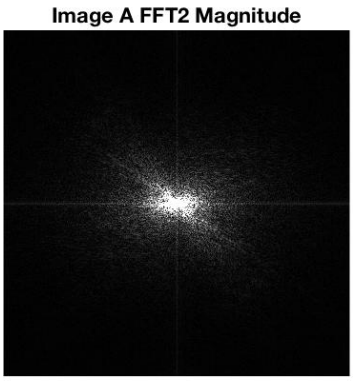


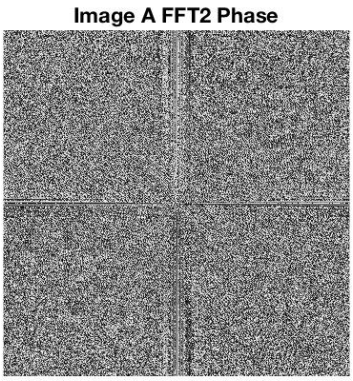
So from the comparison we can see that the narrow the Gaussian is, the wider their DFT is.

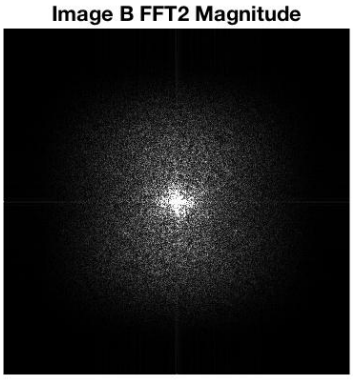
Two original images:

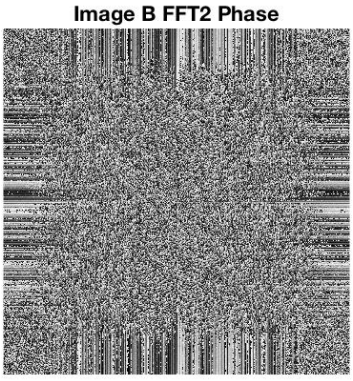


computing the DFT of these two images:









computing a new set of coe cents which have the phase from the first image and the mangitude from the second image,

```
fftD = abs(fftB).*exp(i*angle(fftA));
```

taking the inverse DFT of this combined spectrum to produce a new image.

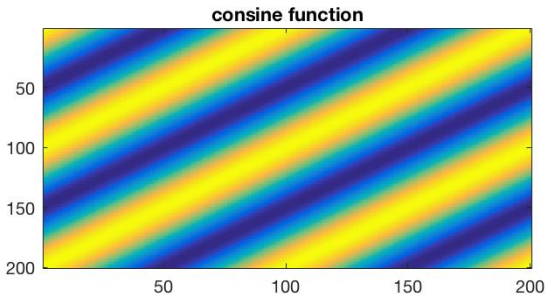
```
imageD = ifft2(fftD);
```

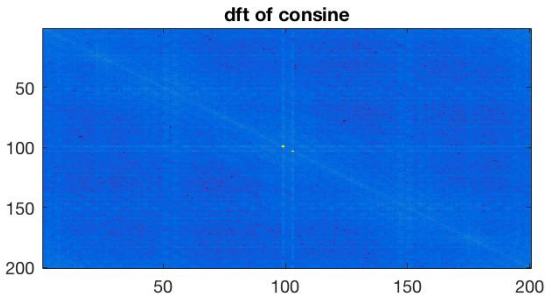
Image D phase from the first image and the mangitude from the second image

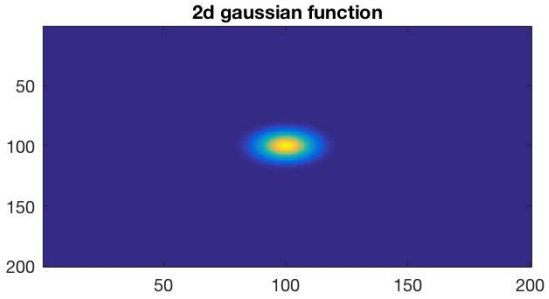
So we can see that phase has more information about image. While phase is about texture.

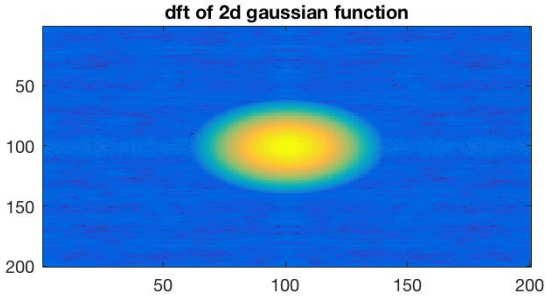
Last part: Fun experiments

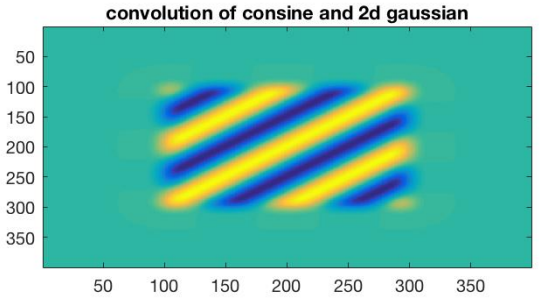
Doing convolution of 2D Gaussian function with sin() function.

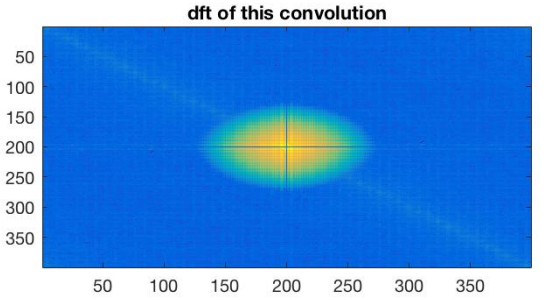

consine function
50
100
150
200
50
100
150
200


dft of consine
50
100
150
200
50
100
150
200


2d gaussian function
50
100
150
200
50
100
150
200

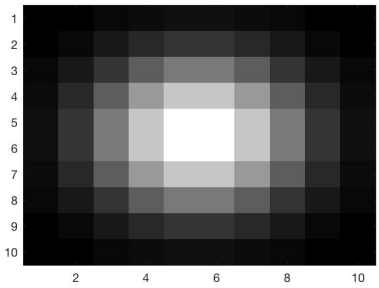

dft of 2d gaussian function
50
100
150
200
50
100
150
200

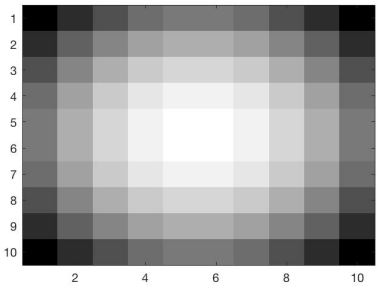

convolution of consine and 2d gaussian
50
100
150
200
250
300
350
50
100
150
200
250
300
350


dft of this convolution
50
100
150
200
250
300
350
50
100
150
200
250
300
350

6. Edge detector

After applying Gaussian to Original image and the filter:





Based on the derivative equation: F'(x) = 1/2* [f(x+dx) – f(x-dx)] /2, we get derivative filters:

-1 0 1 (Horizental)

-2 0 2

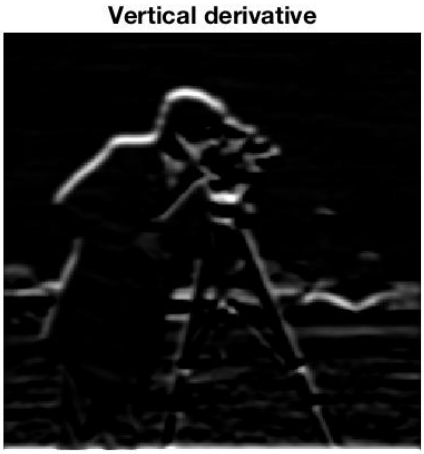
-1 0 1

-1 -2 -1 (Vertical)

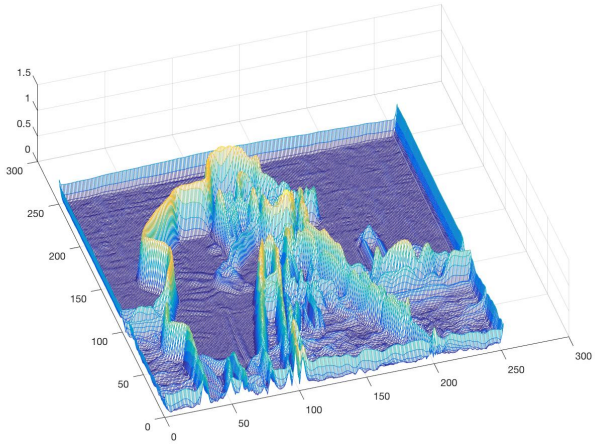
0 0 0

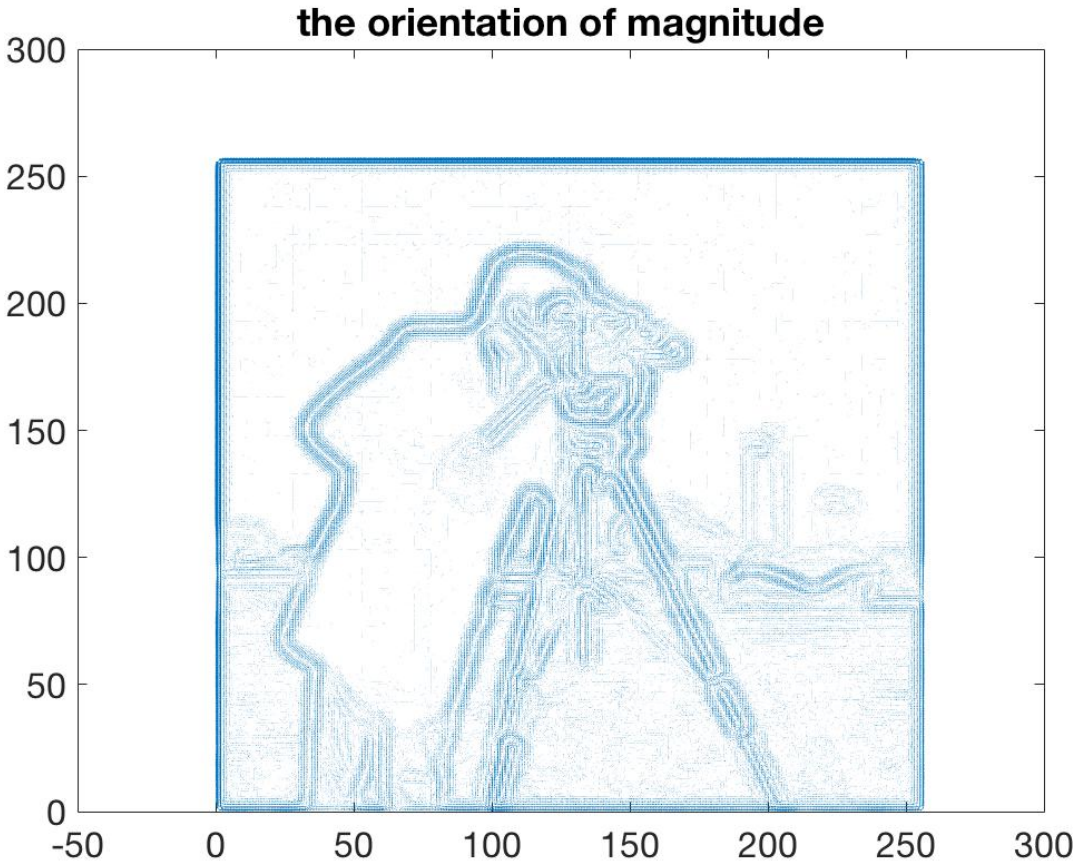
1 2 1

So pixal with higher derivative has bigger value. Following are result of horizontal edge detector(left) and vertical edge detector (right)



Compute the gradient magnitude(left) and orientation (3d figure of magnitude or a little arrow on each pixel right and below).





According to the gradient equation:

$$\nabla f = \frac{df}{dx}\, e_x + \frac{df}{dy}\, e_y + \frac{df}{dz}\, e_z$$

Since there is no z direction. So the gradient is

$$Dx * e_x + Dy * e_y$$

And according to $\tan(\theta) = \frac{df}{dy} / \frac{df}{dx}$. We can calculate the direction of gradient on every pixel: (θ) =Arctan($\frac{df}{dy} / \frac{df}{dx}$) .

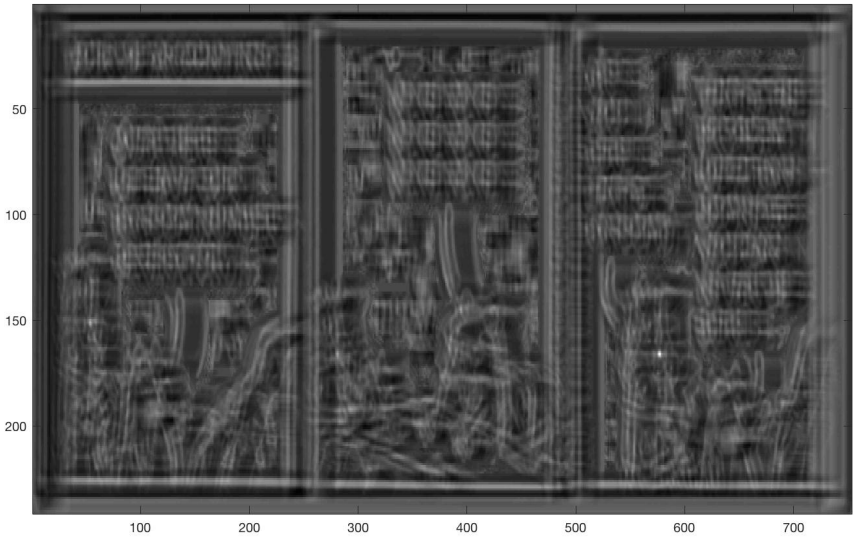
Magnitude: $|\nabla f| = \sqrt{(\frac{df}{dx})^2 + (\frac{df}{dy})^2}$

Then to get the magnitude matrix (image) we need to do above calculation for every pixel on the image.

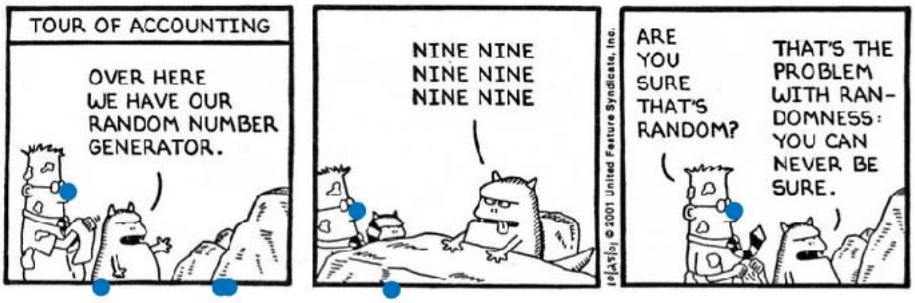
7. **Implement a simple object detector based on correlation.**

First I use the built in method **normxcorr2**. Below is the fist image I detect some little object.

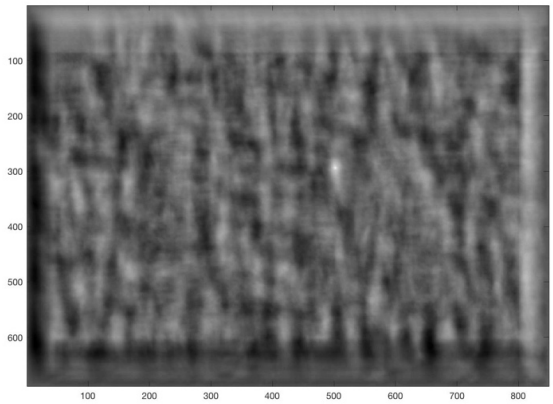
Detect the glass (left)

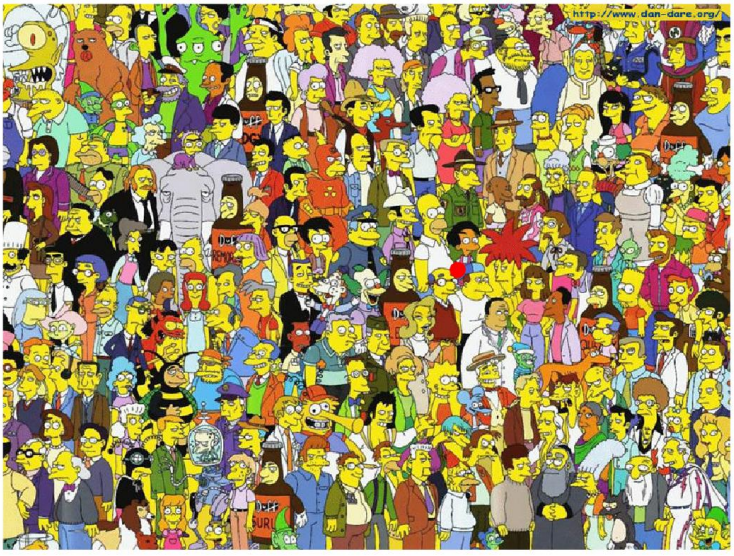


We can see light spot on the correlation map. So these local maxima indicate where there is a close match between the image and template.



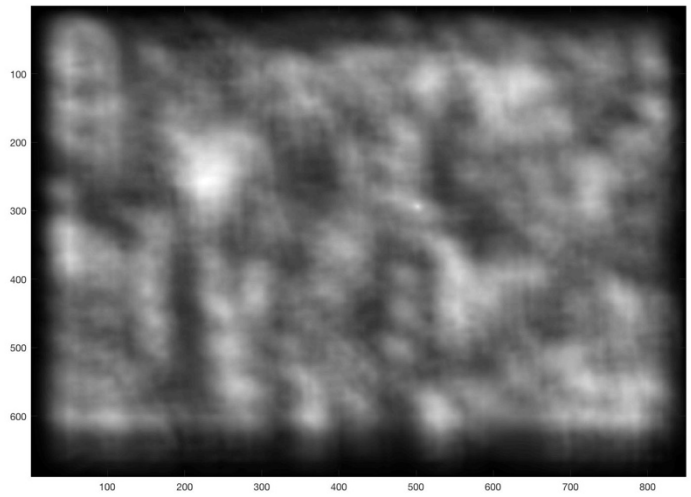
Here I use a Gaussian filter to blur the template first. I use threshold 0.47 for finding the local maxima. Then I find these parameters set give a good performance. The I try to find some big object on original image like the Simpson below. The template(left), the correlation map(right)

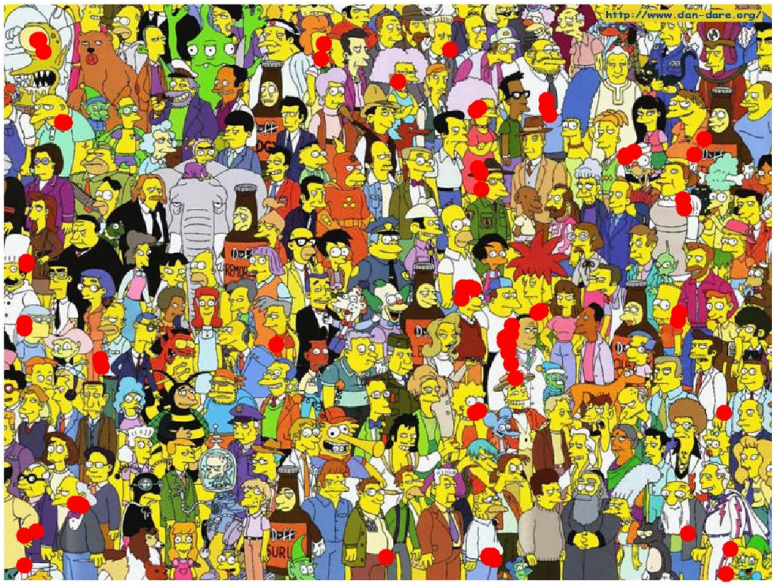




Above are the template, the correlation image, and the point I find on original image. Here I use 0.46 as the threshold. And we can just find Simpson on image.

We can also use Conv2 to calculate, but due to the lack of normalization, the value become very big. We need to use a big threshold.

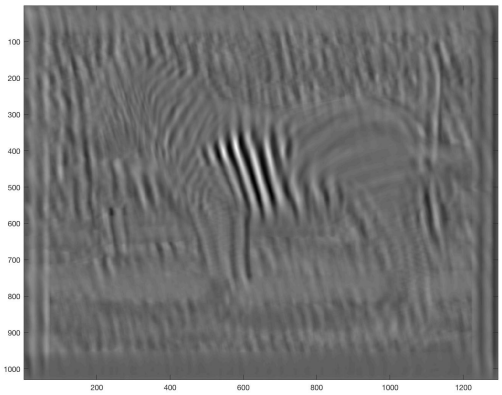




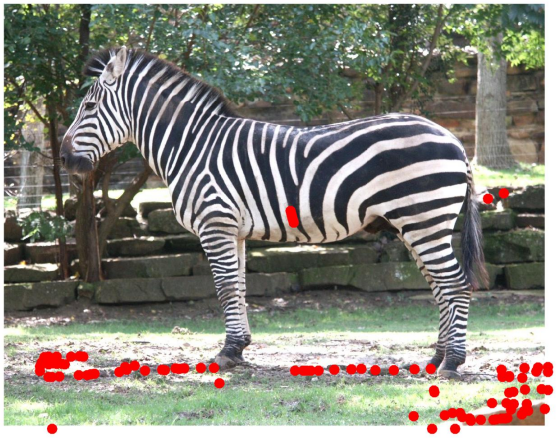
When we choose the right threshold 135000000 in for this template, we can also find some match. In above example. We can find that we have a match over template, we also have some match which are similar to the pattern of our template (yellow with some white). So it actually works.

The detection of pattern.

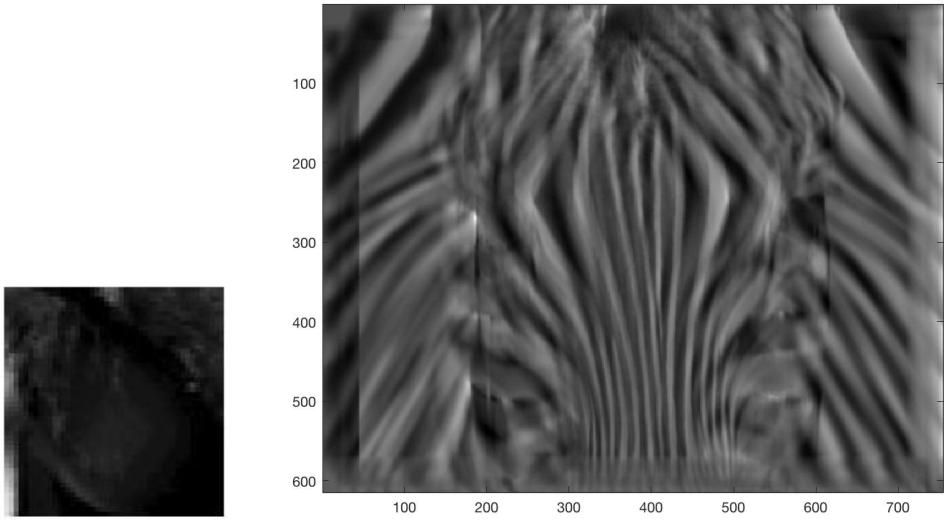
The pattern on zebra that I want to detect.



The normxcorr2 can give a good match(Left). Conv2 can highlight all the similar details, but cannot highlight the template well. And is more blurred. When I use the correlation map from to detect pattern, I can find the pattern if I use right threshold, but the ground which has similar white color, will also be highlighted on correlation map.



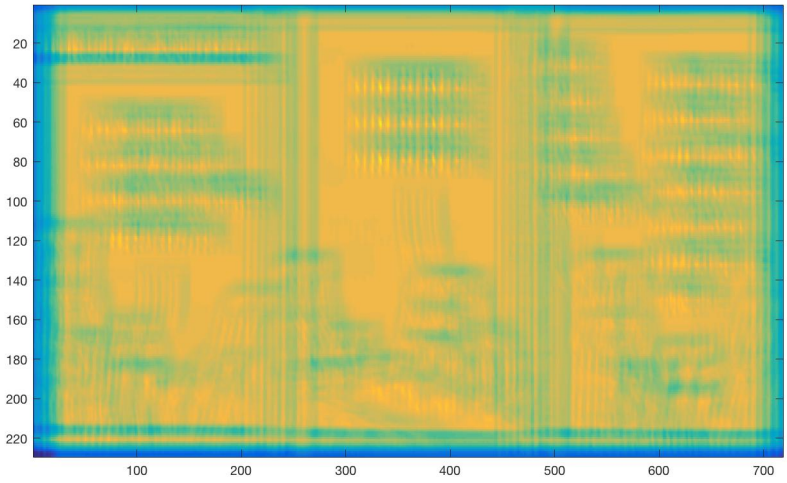
Here I want to find eyes(left) on the graph of face of zebra(right is correlation map of zebra face).



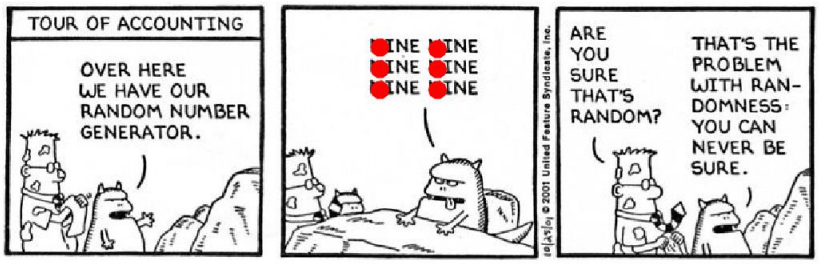
So the correlation cannot find another eye. Even though they are all black. Maybe it is because they have different neighbors. And they are symmetrical, so their horizontal local details are different.

8. compute the sum of squared differences (SSD)

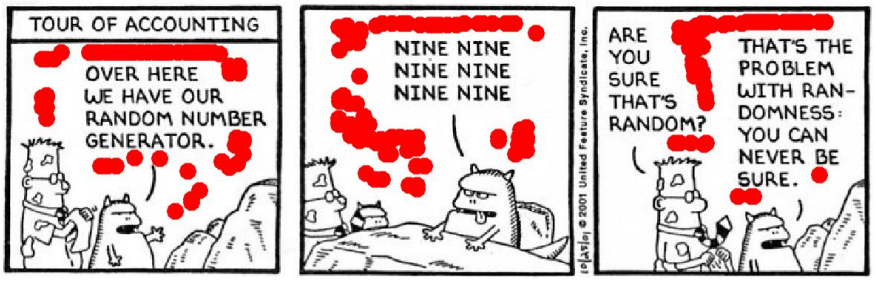
Here the SSD has better detection ability than using conv2.



Above is the plot of correlation map of using SSD method. Here we can clearly see some bright spots on this map, which detects “Nine” template pretty well (bellow).

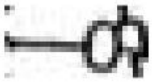


The figure below is the result I get from using conv2.

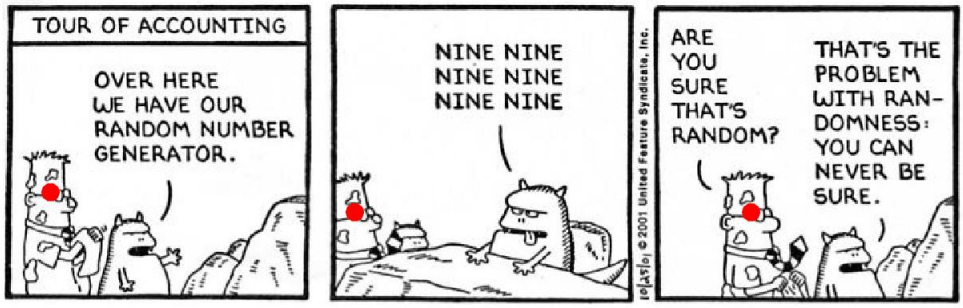


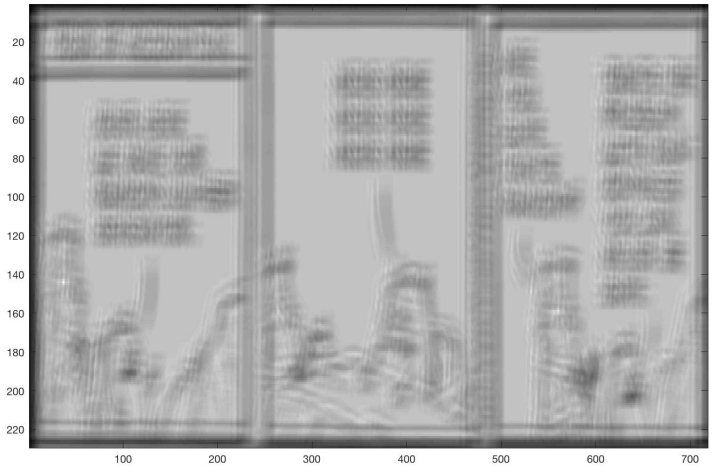
So conv2 totally can not detect glass as it is a template in gray image, because here template and original image has more white(high) value, so detector detect a lot fake points in white background.

It is same for glass template.



If I use original Conv2, it seems totally fails as it only detect white background.



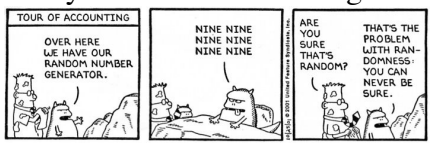


The plot of SSD result. Here we can see clear bright point at matching point.

So the ssd method has a better detecting ability than using conv2 to do correlation.

The analysis of three detector strategy:

Conv2 to do correlation:

Using convolution to do correlation, without normalization, the only way to do normalization is being divided by a value. So the background of the original image is high and flat, like the image we get for 

, when we calculate:

$f * g = \int_{-\infty}^{\infty} f(x)g(y+x)dx$. It is easy to get a very high very even if the patch on image and template do not match. So it cannot perform well.

The SSD to modify correlation:

Let's analyze the algorithm of SSD,

```
IT = conv2(I,T);
Tsquared = sum(sum(T.^2));
Isquared = conv2(I.^2, ones(size(T)));
squareddiff = (Isquared - 2*IT + Tsquared);
```

So here **Tsquared** is like a threshold, it is just a trivial constant over all image. So I think it does not matter too much here. The IT is a convolution of template and image as we do in original conv2, which will give high value when these is a good match, but the difference is that here we apply a negative sign. So matching place will be low. Here Isquared is the crucial part which modify the algorithm a lot. Below I draw the plot for Isquared with some normalization:

So it is like a similar figure which capture the energy information of original image, as it is a square of original image and convolve with a trivial filter (ones matrix) that average the pixel with all other pixel within the template. **So the modification here is that it can remove places which are high and flat from original image**. So it is like what this Isquared do is that it can help highlight place which has a lot of fluctuations (where features usually happen) in original image.

Also this ssd is in the form of (I-T)^2. So it can give the difference between I and T from mathematical perspective.

The normxcorr2 to do correlation:

Algorithms

`normxcorr2` uses the following general procedure , :

1. Calculate cross-correlation in the spatial or the frequency domain, depending on size of images.
2. Calculate local sums by precomputing running sums.
3. Use local sums to normalize the cross-correlation to get correlation coefficients.

The implementation closely follows the formula from :

$$\gamma(u,v) = \frac{\sum_{x,y}\left[f(x,y) - \bar{f}_{u,v}\right]\left[t(x-u,y-v) - \bar{t}\right]}{\left\{\sum_{x,y}\left[f(x,y) - \bar{f}_{u,v}\right]^2 \sum_{x,y}\left[t(x-u,y-v) - \bar{t}\right]^2\right\}^{0.5}}$$

where

- f is the image.
- $\bar{t}$ is the mean of the template
- $\bar{f}_{u,v}$ is the mean of $f(x,y)$ in the region under the template.

So here it uses the cross-correlation, which can get rid of the influence of background, like if the template all are very high values(white), then subtract average will get rid of this influence and only get useful information- the local changes of template and patch. Also, it uses local sums to normalize, so the value can be within a range.