Lab6 - Image Restoration

BY WANG ZIMENG

12011711@mail.sustech.edu.cn

1 Introduction

Image restoration is the operation of transforming the noisy or contaiminated image back into a clear, orginal image. There are many times that a blur would be included : motion blur, noise and camera misfoocus. By taking the reverse step using prior knowledge, image restoration is performed to reduce this blur.

Unlike image enhancement which is designed to emphasizing features to please the reader, removing some noise by sacrificing some resolution, image restoration aims to produce realistic data from scientific point of view. There are many filters operated in this lab. Alpha-trimmed mean filter only remain some middle pixels in a cell. Harmonic and contraharmonic mean filter could solve salt noise and pepper noise well seperately. Adaptive filters with changing cell could fit with pepper and salt noise. Then, using prior knowldege, we could apply inverse filters, Wiener filter and constrained least square mean filter seperately to reduce systematic noise.

Image restoration is widely used in many places that would cause image lose. Some historical images and badly-performed camera image are most of the case.

2 Implementation

2.1 Q6_1_1 and Q6_1_2 : Contraharmonic and harmonic filter

As we can see from the image given, these two images only have pepper noise and salt noise seperately. So, we could apply contraharmonic hand harmonic filter seperately to eliminate that.

Harmonic filter is given by

$$f(x,y) = \frac{mn}{\sum_{(s,t)\in S_{xy}} \frac{1}{g(s,t)}}$$

It is clear that, if we have a salt noise in the cell S_{xy}, then the inverse of salt with high intensity could minimize its affect.

Contraharmonic filter is given by

$$\hat{f}(x,y) = \frac{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q+1}}{\sum_{(s,t)\in S_{xy}} g(s,t)^{Q}}$$

This filter could solve pepper noise greatly since, similarly, the affect of the pepper noise with low intensity will be small if $Q > 0$. At the other side, if we make $Q < 0$, the influence of salt noise with high intensity will be reduced a lot.

```
pad_img = np.pad(img,((s_size,s_size),(s_size,s_size)),'symmetric')

for i in range(h):
    for j in range(w):
        S = pad_img[i:i+2*s_size+1,j:j+2*s_size+1]
        S = np.float32(S.flatten())
        S = S+0.000001
        S_n = pow(S,Q+1)    #numerator
        S_d = pow(S,Q)      #denumoninator

        img[i,j] = S_n.sum()/S_d.sum()
```

Figure 1. Contraharmonic

```
pad_img = np.pad(img,((s_size,s_size),(s_size,s_size)),'symmetric')

for i in range(h):
    for j in range(w):
        S = pad_img[i:i+2*s_size+1,j:j+2*s_size+1]
        S = S.flatten()
        S = 1/S
        img[i,j] = (2*s_size+1)*(2*s_size+1)/S.sum()
```

Figure 2. Harmonic

The code is simple, cell is moving during iteration and we follow the mathmatical instruction in above.

Time and Space Complexity

For this implement, since we only need to keep a storage of a cell with given size, the space complexity is $O(1)$. For a image $m \times n$, the time complexity will be $O(mn)$ since the time consumption is each step is the same and there are $m \times n$ pixels at all.

2.2 Q6_1_3 and Q6_1_4 : Adaptive median filter

Adaptive median filter is a quite engineering filter, so there is not many mathmatics in it. The basic idea is that if the median is the minimam or maximam then it is a noise and we need to use a larger scale of cell to prevent this and if the pixel is mamximam or minimam then we coule use median to replace the noise.

Stage A :

$A_1 = z_{\text{med}} - z_{\text{min}}; \qquad A_2 = z_{\text{med}} - z_{\text{max}}$

if $A_1 > 0$ and $A_2 < 0$: go to stage B

else : increase the window size until it reaches the maximam S_{max}

Stage B:

$B_1 = z_{\text{xy}} - z_{\text{min}}; \qquad B_2 = z_{\text{xy}} - z_{\text{max}}$

if $B_1 > 0$ and $B_2 < 0$: output z_{xy},

else : output z_{med}

Pseudo Code :

```
gmin <- min of S(x,y) of size n*n
gmed <- median
gmax <- max
g <- img[x,y]
s_max <- max size of S

for every iteration:
    if(gmin<gmed<gmax):
        img[x,y] = gmed
        if (gmin<g<gmax):
            img[x,y] = g
     while not (gmin<gmed<gmax and n>=s_max):
```

```
S <- n+1*n+1
if(gmin<gmed<gmax):
  img[x,y] = gmed
  if(gmin<g<gmax):
    img[x,y] = g
```

```
S = pad_img[i-s_size+p_size:i+s_size+p_size+1,j-s_size+p_size:j+s_size+p_size+1]
S = S.flatten()
gmin = S.min()
gmax = S.max()
gmed = np.median(S)
g = img[i,j]
## Stage A
if(gmin<gmed<gmax):
    if (gmin<g<gmax):
        img[i,j] = img[i,j]
    else :
        img[i,j] = gmed
else :  ## Stage B
    while s_size<=s_max:
        s_size = s_size+1
        S = pad_img[i-s_size+p_size:i+s_size+p_size+1,j-s_size+p_size:j+s_size+p_size+1]
        S = S.flatten()
        gmin = S.min()
        gmax = S.max()
        gmed = np.median(S)
        g = img[i,j]
        ### Stage A in Stage B ##
        if(gmin<gmed<gmax):
            if (gmin<g<gmax):
                img[i,j] = img[i,j]

            else :
                img[i,j] = gmed
            s_size = origin_s
            break
        if(s_size==s_max):
            img[i,j] = gmed
            s_size= origin_s
            break
```

Figure 3. Adaptive Median

the code follow the step above. But we check the stage A in a while loop directly instead of going back to another function.

2.3 Q6_2 : Wiener filter

Wiener filter is based on the degradation fuction estimated or tested and with other stochastic

noise. The goal is to find the optimal function $\hat{f}$ with the cost of mean square error :

$$e^2 = E\{(f - \hat{f})^2\}$$

The solution is given by :

$$\hat{F}(\mu,\nu) = \left[\frac{|H(\mu,\nu)|^2}{H(\mu,\nu) \times |H(\mu,\nu)|^2 + S_\eta(\mu,\nu)/S_f(\mu,\nu)}\right] G(\mu,\nu)$$

where H is the estimated degradation function and G is the original image.

In fact ,we could not know the noise data, so we often use this formula for Wiener filter instead :

$$\hat{F}(\mu,\nu) = \left[\frac{|H(\mu,\nu)|^2}{H(\mu,\nu) \times |H(\mu,\nu)|^2 + K}\right] G(\mu,\nu)$$

And we need to select the best K for better performance with the evaluation of SNR.

Code :

```
mid_u = int(h/2)
mid_v = int(w/2)
H = np.zeros_like(img)
for u in range(h):
    for v in range(w):
        d = (u - mid_u) ** 2 + (v - mid_v) ** 2
        H[u,v] = math.exp(-k*(d**(5/6)))

abs_H = abs(H)
img =  abs_H**2/((abs_H**2+K)*H)*img

img = np.fft.ifft2(np.fft.ifftshift(img)).real
img = (img-img.min())/(img.max()-img.min())*255
return img
```

Figure 4. Wiener

Time and Space Complexity

If we ignore the time consumption which is essential time for frequency image processing. The time complexity is $O(mn)$ with image $m \times n$. and the space complexity is $O(1)$ since we just need to constant number of variables.

Q6_2 : Constrianed least square

If we express the result in matrix form, we would have :

$$\boldsymbol{g} = H\boldsymbol{f} + \boldsymbol{\eta}$$

where g, η, and f are vectors of size $MN \times 1$, and H is a degradation matrix of size $MN \times MN$.

Skip some mathmatical derivation ,we could get

$$C = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\nabla^2 f(x,y)]^2$$

with constrain :

$$\|\boldsymbol{g} - H\hat{\boldsymbol{f}}\|^2 = \|\boldsymbol{\eta}\|^2$$

Then we could get

$$\hat{F}(\mu,\nu) = \left[\frac{H^*(\mu,\nu)}{|H(\mu,\nu)|^2 + \gamma|P(\mu,\nu)|^2}\right] G(\mu,\nu)$$

where $P(\mu,\nu)$ is the $p(x,y)$ in freqency domain, $p(x,y)$ is

$$p(x,y) = \begin{pmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{pmatrix}$$

Code :

```python
def CLS_filter(img,gamma,a=0.1,b=0.1,T=1):
    h,w = img.shape
    img = np.fft.fft2(img)
    img = np.fft.fftshift(img)

    mid_u = int(h/2)
    mid_v = int(w/2)
    H = np.zeros_like(img)
    for x in range(h):
        for y in range(w):
            u = x - mid_u
            v = y - mid_v
            c = u*a+v*b
            if (c==0):
                H[x,y] = 1
                continue
            H[x,y] = T*math.sin(math.pi*(c))*math.e**(-math.pi*(c)*1j)/(math.pi*(c))

    abs_H = abs(H)
    P = np.zeros_like(img)
    laplace = np.array([[0, -1, 0], [-1, 4, -1], [0, -1, 0]], dtype=np.float)
    P[h // 2 - 1: h // 2 + 2, w // 2 - 1: w // 2 + 2] = laplace
    P = np.fft.fftshift(np.fft.fft2(P))
    img =  np.conj(H)/(abs_H**2+gamma*abs(P)**2)*img

    img = np.fft.ifft2(np.fft.ifftshift(img)).real
    img = (img-img.min())/(img.max()-img.min())*255
```

Figure 5. CLS

Time and Space Complexity

For this code, still, we only need to calculate the degradation function and have a time complexity $O(mn)$ regardless of fouier transformation. For space complexity, we have $O(mn)$ also since we

need to calculate the laplace kernal.

2.4 Q6_3

The way we process the Q6_3 is excatly the same except for the degradation function H. For Q6_2 we have

$$H = e^{-k(\mu^2+\nu^2)^{5/6}}$$

However, for the linear motion we have :

$$H = \frac{T}{\pi(\mu a + \nu b)} \sin(\pi(\mu a + \nu b)) e^{-j\pi(\mu a + \nu b)}$$

where $x_0(t) = at/T$ and $y_0(t) = bt/T$

3 Analysis

3.1 Q6_1_1 and Q6_1_2

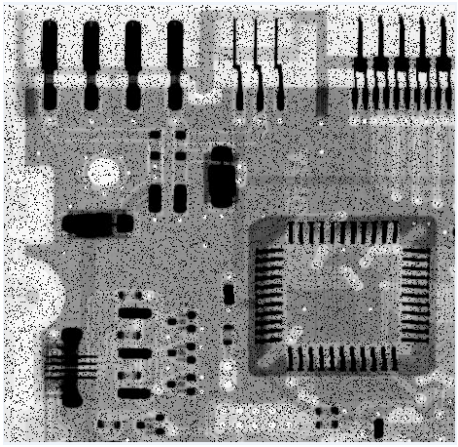

Figure 6.

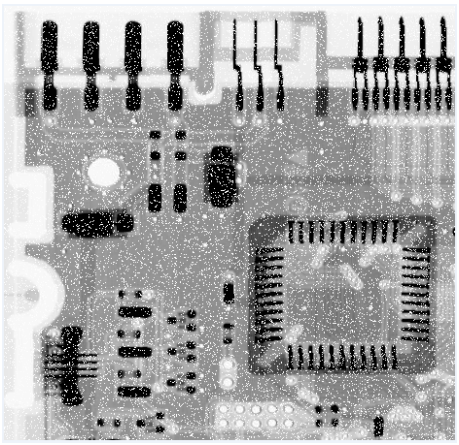

Figure 7.

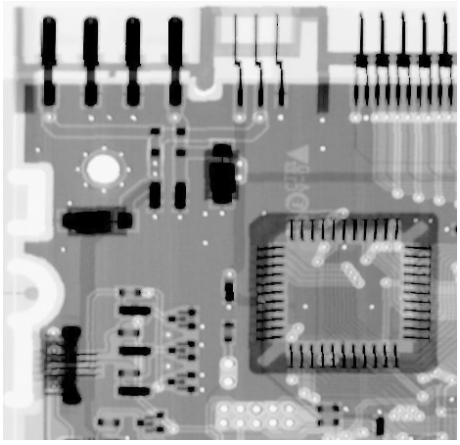

Figure 8. Q6_1_1

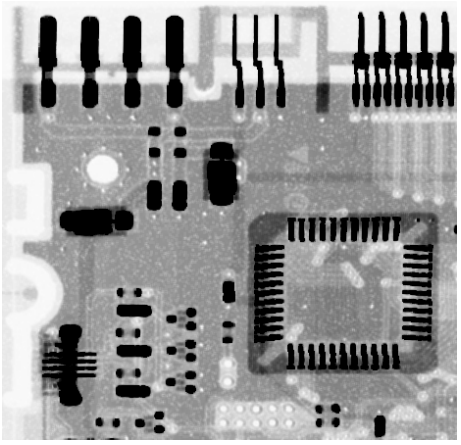

Figure 9. Q6_1_2

According to the previous derivation, we know that Contraharmonic Mean Filter filters out pepper noise when Q>0. Here, choosing Q=1.5 has achieved good results. Both with a cell size of 3.

3.2 Q6_1_3

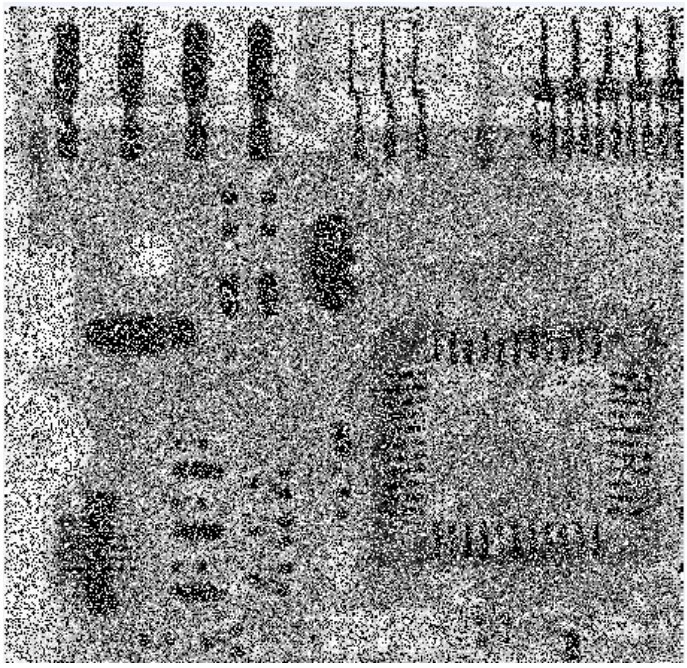

Figure 10.

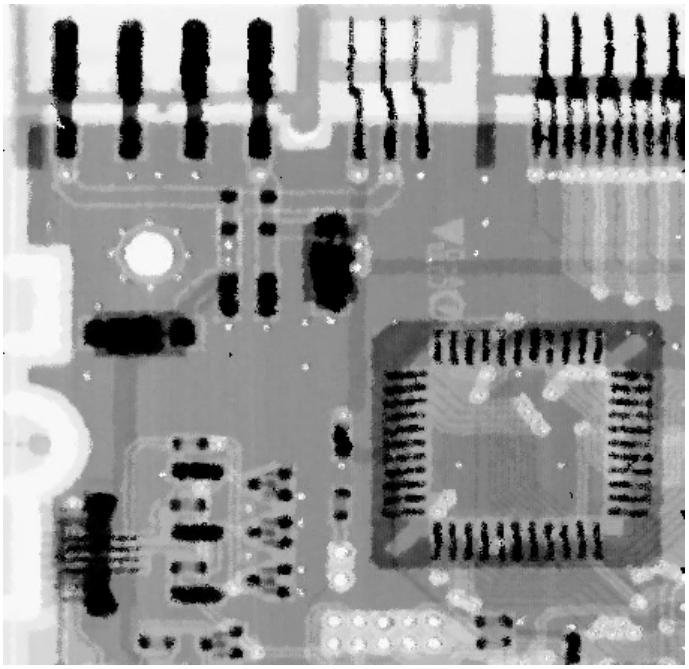

Figure 11. Adaptive

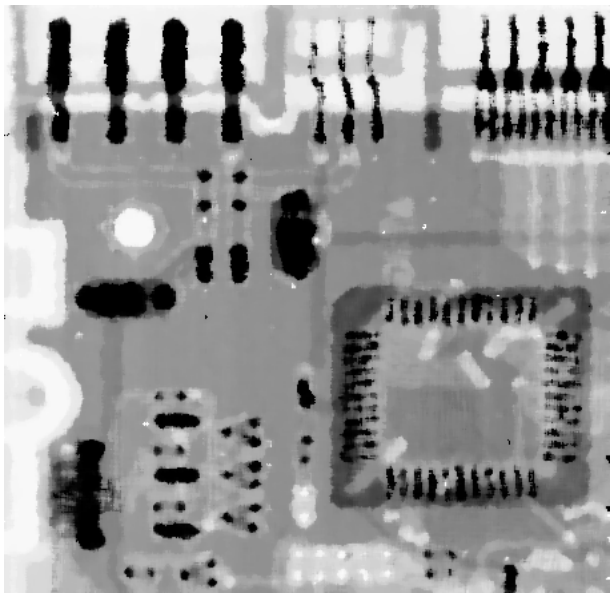

Figure 12. Median

This question uses an adaptive median filter with a minimum window of 3 and a maximum window of 7, and with comaparation of Median with cell 7. It can be seen from the results that the median filter have a relatively not bad result since some of the details are missed and it seems vague.

3.3 Q6_1_4

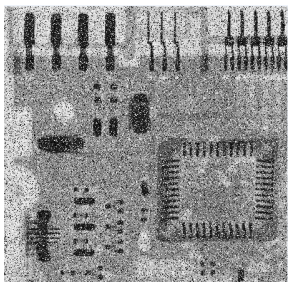

Figure 13.

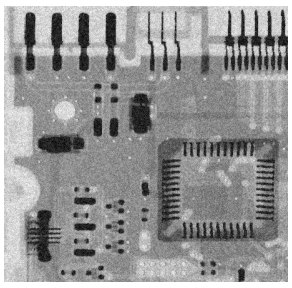

Figure 14. Adaptive

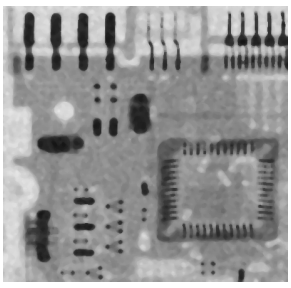

Figure 15. Median

In comparasion, I use the median filter after median filter which produces a recognizable figure. At

the same time, the adaptive median filter have a better result thoght it still have some of the noises.

3.4 Q6_2

Figure 16. Inverse with LPF60

Figure 17. Wiener K=0.0005

Figure 18.

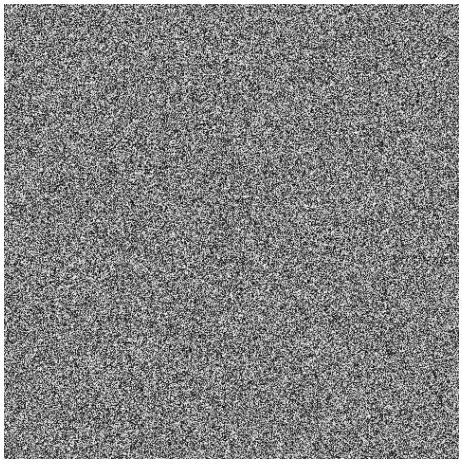

Figure 19. Inverse Without LPF300

First, I use the full inverse filter, and we can find that the image is unrecognizable. This is because the degraded model have some 0s in some place and will affect the effect a lot. But after the LPF, the effect is much better.

Figure 20. D=30

Figure 21. D=40

Figure 22. D=50

Figure 23. D=60

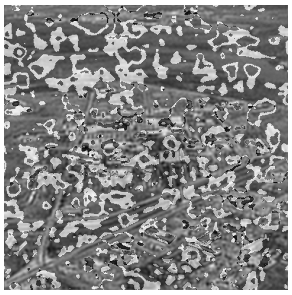

Figure 24. D=70

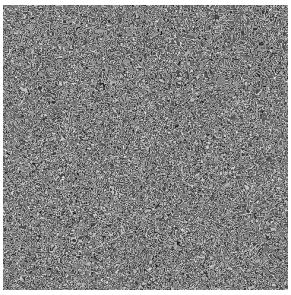

Figure 25. D=80

Then I use limit inverse filter. It can be seen from the results that after adding the Butterworth low-pass filter, the image can be recognized, and the best effect is when the cut-off radius is about 60.

Figure 26. K=0.0001

Figure 27. K=0.0005

Figure 28. K=0.001

Figure 29. K=0.005

Figure 30. K=0.01

Figure 31. K=0.1

For the Wiener filter, we have a optimal K of 0.0005

3.5 Q6_3 Bonus

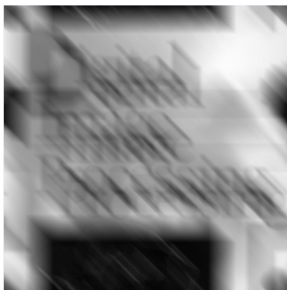

Figure 32. Q6_3_1

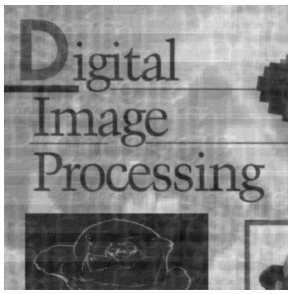

Figure 33. Wiener

Figure 34. CLS

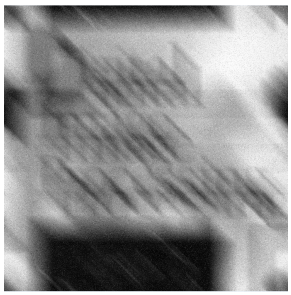

Figure 35. Q6_3_2

Figure 36. Wiener

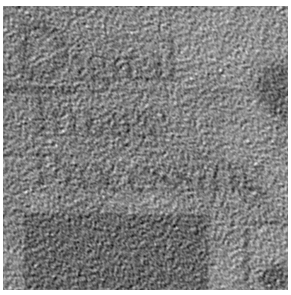

Figure 37. CLS

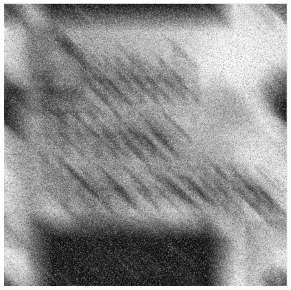

Figure 38.

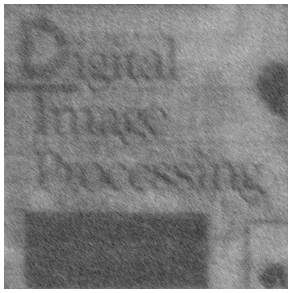

Figure 39. Wiener

Figure 40. CLS

I used Wiener filter and Constrained Least Square filter to restore this three motion-blurred image. As shown in the image, when the noise is relatively small, the restoration have a perfect effect. However, as the intensity of noise going up, the restoration could only perform a recognizable image regardless of what method we use.

4 Conclusion and Prospect

In conlusion, there are many ways of image restoration in this lab. Including Harmonic filter that could only solve salt noise, Contraharmonic filter that could solve pepper or salt noise at a time, Adaptive filter and Alpha trim filter that could reduce salt and pepper noise by itself and degradtion function to reverse the image loss.

Nowadays, as the machine learning going hot, there are also many ways to restore a image or even

color a gray image through deep learning. Throgh deep learning, the neural network learn the content and feature from the input sample and add the loss or generate the new image by itself.

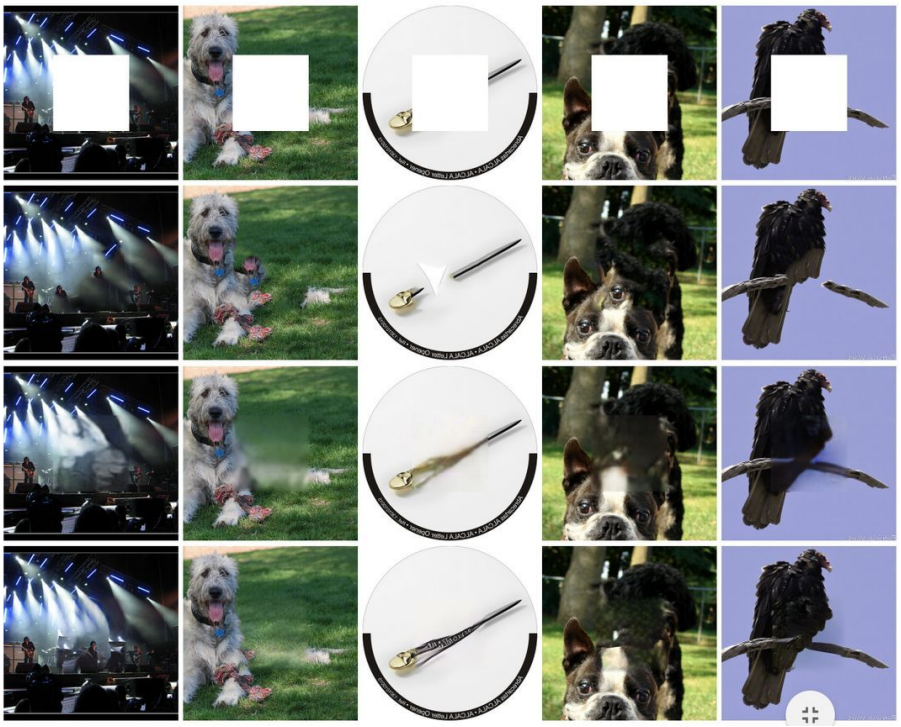

Figure 41.