信 息 科 学 与 技 术 学 院
SCHOOL OF INFORMATION SCIENCE&TECHNOLOGY

西北大学
Northwest University

计算机视觉

彭盛霖

西北大学信息学院

pengshenglin@nwu.edu.cn

8 图像修复

8 图像修复

高斯噪声

```python
def normalize(mask,cut=True):
    if cut: return np.clip(mask,0,255)/255.0
    return (mask - mask.min()) / (mask.max() - mask.min())
def add_gaussian_noise(img, mu=0, sigma=25):
    """
    add gaussian noise for image
    param:  img:        input image, dtype=uint8
    param:  mean:       noise mean
    param:  sigma:      noise sigma
    return: image_out: image with gaussian noise
    """
    img=np.expand_dims(img,axis=-1) if img.ndim==2 else img
    new_img=np.zeros(img.shape)
    for i in range(img.shape[2]):
        image = np.array(img[:,:,i], dtype=float)
        noise = np.random.normal(mu, sigma, image.shape)
        # print(np.mean(noise),np.std(noise))
        new_img[:,:,i] = normalize(image + noise)*255
    if img.ndim==2: new_img=np.squeeze(new_img,-1)
    return np.uint8(new_img)
```

8 图像修复

瑞利噪声

```python
def add_rayleigh_noise(img, mu=0, sigma=25):
    """
    add gaussian noise for image
    param:  img:        input image, dtype=uint8
    param:  mean:       noise mean
    param:  sigma:      noise sigma
    return: image_out: image with gaussian noise
    """
    img=np.expand_dims(img,axis=-1) if img.ndim==2 else img
    new_img=np.zeros(img.shape)
    for i in range(img.shape[2]):
        image = np.array(img[:,:,i], dtype=float)
        noise = np.random.rayleigh(scale=sigma, size=image.shape)
        noise = noise/np.std(noise)*sigma
        noise = noise-np.mean(noise)+mu
        # print(np.mean(noise),np.std(noise))
        new_img[:,:,i] = normalize(image + noise)*255
    if img.ndim==2: new_img=np.squeeze(new_img,-1)
    return np.uint8(new_img)
```

8 图像修复

伽玛噪声

```
def add_gamma_noise(img,mu=0, sigma=25):
    img=np.expand_dims(img,axis=-1) if img.ndim==2 else img
    new_img=np.zeros(img.shape)
    for i in range(img.shape[2]):
        image = np.array(img[:,:,i], dtype=float)
        a = 2 * mu - np.sqrt(12 * sigma)
        b = 2 * mu + np.sqrt(12 * sigma)
        noise = np.random.uniform(a, b, image.shape)
        # showGrayHist(noise)
        # print(np.mean(noise),np.std(noise))
        # print(b*scale,b**0.5*scale)
        new_img[:,:,i] = normalize(image + noise)*255
    if img.ndim==2: new_img=np.squeeze(new_img,-1)
    return np.uint8(new_img)
```

8 图像修复

椒盐噪声

```python
def add_salt_pepper(img, ps=0.05, pp=0.05):
    """
    add salt pepper noise to image
    param: img: input image, uint8 [0, 255]
    param: ps:  probability of salt noise, which is white noise, default is 0.01
    param: pp:  probability of peper noise, which is black noise, default is 0.01
    return image with salt pepper noise, [0, 255]
    """
    img=np.expand_dims(img,axis=-1) if img.ndim==2 else img
    new_img=np.zeros(img.shape)
    h, w = img.shape[:2]
    mask = np.random.choice((0, 0.5, 1), size=(h, w), p=[pp, (1-ps-pp), ps])
    img_out= img
    img_out[mask==1] = 255
    img_out[mask==0] = 0
    new_img=img_out
    if img.ndim==2: new_img=np.squeeze(new_img,-1)
    return np.uint8(new_img)
```

8 图像修复

几何均值滤波

```python
def GeometricMeanOperator(roi):
    roi = roi.astype(np.float64)
    p = np.prod(roi)
    return p ** (1 / (roi.shape[0] * roi.shape[1]))
def GeometricMeanAlogrithm(image):
    # 几何均值滤波
    new_image = np.zeros(image.shape)
    image = cv2.copyMakeBorder(image, 1, 1, 1, 1, cv2.BORDER_DEFAULT)
    for i in range(1, image.shape[0] - 1):
        for j in range(1, image.shape[1] - 1):
            new_image[i - 1, j - 1] = GeometricMeanOperator(image[i - 1:i + 2, j - 1:j + 2])
    new_image = (new_image - np.min(image)) * (255 / np.max(image))
    return new_image.astype(np.uint8)
def rgbGemotriccMean(image):
    r,g,b = cv2.split(image)
    r = GeometricMeanAlogrithm(r)
    g = GeometricMeanAlogrithm(g)
    b = GeometricMeanAlogrithm(b)
    return cv2.merge([r,g,b])
```

谐波均值滤波

```python
def HarmonicMeanOperator(roi):
    roi = roi.astype(np.float64)
    if 0 in roi:
        roi = 0
    else:
        roi = scipy.stats.hmean(roi.reshape(-1))
    return roi
def HarmonicMeanAlogrithm(image):
    # 谐波均值滤波
    new_image = np.zeros(image.shape)
    image = cv2.copyMakeBorder(image,1,1,1,1,cv2.BORDER_DEFAULT)
    for i in range(1,image.shape[0]-1):
        for j in range(1,image.shape[1]-1):
            new_image[i-1,j-1] =HarmonicMeanOperator(image[i-1:i+2,j-1:j+2])
    new_image = (new_image-np.min(image))*(255/np.max(image))
    return new_image.astype(np.uint8)
def rgbHarmonicMean(image):
    r,g,b = cv2.split(image)
    r = HarmonicMeanAlogrithm(r)
    g = HarmonicMeanAlogrithm(g)
    b = HarmonicMeanAlogrithm(b)
    return cv2.merge([r,g,b])
```

逆谐波均值滤波

```python
def Contra_harmonicMeanOperator(roi,q):
    roi = roi.astype(np.float64)
    return np.mean((roi)**(q+1))/np.mean((roi)**(q))
def Contra_harmonicMeanAlogrithm(image,q):
    # 逆谐波均值滤波
    new_image = np.zeros(image.shape)
    image = cv2.copyMakeBorder(image,1,1,1,1,cv2.BORDER_DEFAULT)
    for i in range(1,image.shape[0]-1):
        for j in range(1,image.shape[1]-1):
            new_image[i-1,j-1] = Contra_harmonicMeanOperator(image[i-1:i+2,j-1:j+2],q)
    new_image = (new_image-np.min(image))*(255/np.max(image))
    return new_image.astype(np.uint8)
def rgbContra_harmonicMean(image,q):
    r,g,b = cv2.split(image)
    r = Contra_harmonicMeanAlogrithm(r,q)
    g = Contra_harmonicMeanAlogrithm(g,q)
    b = Contra_harmonicMeanAlogrithm(b,q)
    return cv2.merge([r,g,b])
```

图像退化恢复

```python
# 仿真运动模糊
def get_motion_dsf(image_size, motion_angle, motion_dis):
    """
    Get motion PSF
    param: image_size: input image shape
    param: motion_angle: blur motion angle
    param: motion_dis: blur distant, the greater value, more blurred
    return normalize PSF
    """
    PSF = np.zeros(image_size)  # 点扩散函数
    x_center = (image_size[0] - 1) / 2
    y_center = (image_size[1] - 1) / 2
    sin_val = np.sin(motion_angle * np.pi / 180)
    cos_val = np.cos(motion_angle * np.pi / 180)
    # 将对应角度上motion_dis个点置成1
    for i in range(motion_dis):
        x_offset = round(sin_val * i)
        y_offset = round(cos_val * i)
        PSF[int(x_center - x_offset), int(y_center + y_offset)] = 1
    return PSF / PSF.sum()    # 归一化
```

图像退化恢复

```python
# 仿真湍流模糊
def cal_distance(pa, pb):# 欧拉距离计算函数的定义
    return np.sqrt((pa[0] - pb[0]) ** 2 + (pa[1] - pb[1]) ** 2)
def get_turbulence_dsf(image_size, k=0.1):
    """
    return normalize PSF
    """
    # print()
    center_point = tuple(map(lambda x: (x - 1) / 2, image_size))  # 中心点
    pos_matrix = np.mgrid[0:image_size[0],0:image_size[1]]
    dis = cal_distance(pos_matrix, center_point)
    # PSF_fft = dis
    PSF_fft = np.exp(-k* dis**(5/6))# 点扩散函数 fft
    PSF = np.fft.ifft2(PSF_fft) # image FFT multiply PSF FFT
    PSF = np.abs(np.fft.ifftshift(PSF))
    return PSF / PSF.sum()    # 归一化
```

图像退化恢复

```python
# 对图片进行模糊
def make_blurred(img, PSF, eps):
    """
    blurred image with PSF
    param: input: input image
    param: PSF: input PSF mask
    param: eps: epsilon, very small value, to make sure not divided or multiplied by zero
    return blurred image
    """
    img=np.expand_dims(img,axis=-1) if img.ndim==2 else img
    new_img=np.zeros(img.shape)
    for i in range(img.shape[2]):
        input_fft = np.fft.fft2(img[:,:,i])                 #  image FFT
        PSF_fft = np.fft.fft2(PSF)+ eps                 # PSF FFT plus epsilon
        blurred = np.fft.ifft2(input_fft * PSF_fft) # image FFT multiply PSF FFT
        blurred = np.abs(np.fft.ifftshift(blurred))
        new_img[:,:,i]=blurred
    if img.ndim==2: new_img=np.squeeze(new_img,-1)
    return np.uint8(new_img)
```

8 图像修复

图像退化恢复

```python
def inverse_filter(img, PSF, eps):
    """
    inverse filter using FFT to denoise
    param: input: input image
    param: PSF: known PSF
    param: eps: epsilon
    """
    img=np.expand_dims(img,axis=-1) if img.ndim==2 else img
    new_img=np.zeros(img.shape)
    for i in range(img.shape[2]):
        input_fft = np.fft.fft2(img[:,:,i])                    #  image FFT
        PSF_fft = np.fft.fft2(PSF) + eps                       #噪声功率，这是已知的，考虑epsilon
        result = np.fft.ifft2(input_fft / PSF_fft)             #计算F(u,v)的傅里叶反变换
        result = np.abs(np.fft.ifftshift(result))
        new_img[:,:,i]=result
    if img.ndim==2: new_img=np.squeeze(new_img,-1)
    return np.uint8(new_img)
```

图像退化恢复

```python
def wiener_filter(img, PSF, eps, K=0.01):
    """
    wiener filter for image denoise
    param: input: input image
    param: PSF: input the PSF mask
    param: eps: epsilon
    param: K=0.01: K value for wiener fuction
    return image after wiener filter
    """
    img=np.expand_dims(img,axis=-1) if img.ndim==2 else img
    new_img=np.zeros(img.shape)
    for i in range(img.shape[2]):
        input_fft = np.fft.fft2(img[:,:,i])
        PSF_fft = np.fft.fft2(PSF) + eps
        PSF_fft_1 = np.conj(PSF_fft) / (np.abs(PSF_fft)**2 + K)
        result = np.fft.ifft2(input_fft * PSF_fft_1)
        result = np.abs(np.fft.ifftshift(result))
        new_img[:,:,i]=result
    if img.ndim==2: new_img=np.squeeze(new_img,-1)
    return np.uint8(new_img)
```

8 图像修复

图像退化恢复

```
# 显示原图像
plt.figure(1, figsize=(6, 6))
plt.title("Original Image"), plt.imshow(image[...,[2,1,0]], 'gray')
plt.xticks([]), plt.yticks([])



# 进行模糊处理
# PSF = get_motion_dsf(image.shape[:2], -50, 100)
PSF = get_turbulence_dsf(image.shape[:2])
spectrum_show(PSF,title='PSF Image')# 模糊PSF与谱
blurred = make_blurred(image, PSF, 1e-3)
plt.figure(2, figsize=(8, 8))
plt.subplot(231), plt.imshow(blurred[...,[2,1,0]], 'gray'), plt.title("blurred")
plt.xticks([]), plt.yticks([])

# 逆滤波
result = inverse_filter(blurred, PSF, 1e-3)
plt.subplot(232), plt.imshow(result[...,[2,1,0]], 'gray'), plt.title("inverse deblurred")
plt.xticks([]), plt.yticks([])

# 维纳滤波
result = wiener_filter(blurred, PSF, 1e-3)
plt.subplot(233), plt.imshow(result[...,[2,1,0]], 'gray'), plt.title("wiener deblurred(k=0.01)")
plt.xticks([]), plt.yticks([])

# 添加噪声,standard_normal产生随机的函数
blurred_noisy = np.uint8(blurred + 0.1 * blurred.std() * np.random.standard_normal(blurred.shape))

# 显示添加噪声且模糊的图像
plt.subplot(234), plt.imshow(blurred_noisy[...,[2,1,0]], 'gray'), plt.title("blurred & noisied")
plt.xticks([]), plt.yticks([])

# 对添加噪声的图像进行逆滤波
result = inverse_filter(blurred_noisy, PSF,  0.1+1e-3)
plt.subplot(235), plt.imshow(result[...,[2,1,0]], 'gray'), plt.title("inverse deblurred")
plt.xticks([]), plt.yticks([])

# 对添加噪声的图像进行维纳滤波
result = wiener_filter(blurred_noisy, PSF,  0.1+1e-3)
plt.subplot(236), plt.imshow(result[...,[2,1,0]], 'gray'), plt.title("wiener deblurred(k=0.01)")
plt.xticks([]), plt.yticks([])

plt.tight_layout()
plt.savefig("out.jpg", format='jpg',bbox_inches = 'tight', dpi=96)
plt.show()
```

8 图像修复

图像修补 BSCB

```python
def normalize(mask,cut=False):
    if cut: return np.clip(mask,0,255)/255.0
    return (mask - mask.min()) / (mask.max() - mask.min())
#BSCB https://www.cnblogs.com/jgg54335/p/14561720.html
def BSCB_inpaint(pic_array,mask=None,epsilon=0.1,inpaint_iters=6,anidiffuse_iters=6,delta_ts=0.02,
                 sensitivites=100,diffuse_coef=1):
    # BSCB算法
    pic_copy = pic_array.copy()
    epsilon2=epsilon*epsilon
    pic_copy_ = pic_array.copy()
    for i in range(anidiffuse_iters):#执行各向异性扩散
        dx_dy=np.gradient(pic_copy)#求梯度
        grad_norm=(dx_dy[0]**2+dx_dy[1]**2+epsilon2)**0.5 #epsilon 防止除以0
        if diffuse_coef == 0:
            diffuse_coefs=np.exp(-grad_norm/sensitivites)
        else:
            diffuse_coefs=1 / (1 + grad_norm/sensitivites)
        dxx=np.gradient(dx_dy[0],axis=0)
        dyy=np.gradient(dx_dy[1],axis=1)
        laplacian=(dxx+dyy)#/grad_norm
        if not mask is None:diffuse_coefs=diffuse_coefs*mask
        pic_copy = pic_copy + diffuse_coefs * laplacian
    for i in range(inpaint_iters):#执行修补
        dx_dy=np.gradient(pic_copy)#求梯度
        grad_norm=(dx_dy[0]**2+dx_dy[1]**2+epsilon2)**0.5 #epsilon 防止除以0
        dxx=np.gradient(dx_dy[0],axis=0)
        dyy=np.gradient(dx_dy[1],axis=1)
        laplacian=(dxx+dyy)
        dx_dy_=np.gradient(laplacian)
        if not mask is None:delta_ts=delta_ts*mask
        delta_ts=delta_ts*(grad_norm>0)
        pic_copy = pic_copy - delta_ts * (-dx_dy[0]*dx_dy_[0]+dx_dy[1]*dx_dy_[1])/grad_norm
    pic_new=pic_array.copy()
    pic_new[1:-1,1:-1]=pic_copy[1:-1,1:-1] # 更新
    return pic_new
def gen_pic_with_mask(mask,origin_pic):
    origin_pic[mask==1.0]=128
    return origin_pic
```

8 图像修复

图像修补 BSCB

```
img= cv2.imread("dog_defiled.jpg")
# mask = (img[:,:,2]>220) & (img[:,:,0]<100)
#图片二值化处理，把[0, 0, 200]~[70, 70, 255]以外的颜色变成0
thresh = cv2.inRange(img, np.array([0, 0, 200]), np.array([70, 70, 255]))
#创建形状和尺寸的结构元素
kernel = np.ones((3, 3), np.uint8)
#扩张待修复区域
mask = cv2.dilate(thresh, kernel, iterations=1)/255.0
spectrum_show(mask)
# print(mask.max(),mask.min())
pic=gen_pic_with_mask(mask,img)
# pic= cv2.imread("cat_noise.jpg") #测试去噪
# mask = None #测试去噪
epsilon = 0.1
inpaint_iters=6
anidiffuse_iters= 6
delta_ts = 0.2
sensitivites = 100
diffuse_coef = 1

epochs=201
pic = (pic/255.0).astype(np.float)
pic_copy = np.zeros(pic.shape)
for epoch in range(epochs):
    # 每epoch次显示一次数据，保存一次数据
    if epoch % 40 ==0:
        print('epoch,当前的循环次数：',epoch, np.abs(pic- pic_copy).max())
        # spectrum_show(np.abs(pic- pic_copy))
        cv2.imwrite("dog_filed"+str(epoch)+'.jpg',img=np.uint8(pic*255))
    pic_copy = pic.copy()
    if epoch<epochs-1:
        for i in range(3):
            pic[:,:,i] =
BSCB_inpaint(pic_copy[:,:,i],mask,epsilon,inpaint_iters,anidiffuse_iters,delta_ts,sensitivites,diffuse_coef)
pic=np.uint8(pic*255)
spectrum_show(pic)
```

8 图像修复

图像修补 TV

```python
def normalize(mask,cut=False):
    if cut: return np.clip(mask,0,255)/255.0
    return (mask - mask.min()) / (mask.max() - mask.min())
#TV https://www.cnblogs.com/hxjbc/p/6675901.html
def tv_inpaint(pic_array,mask=None,epsilon=0.1,dt=0.1,lambda_=0.1,withCCD=True):
    # tv算法
    pic_copy = pic_array.copy()
    epsilon2=epsilon*epsilon
    #求梯度
    dx_dy=np.gradient(pic_copy)
    dx_dy=dx_dy/(dx_dy[0]**2+dx_dy[1]**2+epsilon2)**0.5 #epsilon 防止除以0
    #求散度 divergence(zx,zy) =zx_x + zy_y
    dxx=np.gradient(dx_dy[0],axis=0)
    dyy=np.gradient(dx_dy[1],axis=1)
    div=dxx+dyy
    if withCCD: #是否曲率驱动
        k=np.abs(div)
        # if not mask is None: k=k*mask
        k=k/np.max(k)/dt
        k=0.3+0.7*k**0.1
        div=k*div
    if not mask is None: dt=dt*mask
    #迭代求解 I(n+1)=I(n)+ dt * div(grad(I(n))/abs(grad(I(n)))) - lambda*(I(n)-I(0))
    pic_copy = pic_copy + dt * div - lambda_*(pic_copy - pic_array)
    # pic_copy = normalize(pic_copy)
    # print(np.abs(pic_copy- pic_array).max())
    pic_new=pic_array.copy()
    pic_new[1:-1,1:-1]=pic_copy[1:-1,1:-1] # 更新
    return pic_new
def gen_pic_with_mask(mask,origin_pic):
    origin_pic[mask==1.0]=128
    return origin_pic
```

8 图像修复

图像修补 TV

```
img= cv2.imread("dog_defiled.jpg")
# mask = (img[:,:,2]>220) & (img[:,:,0]<100)
#图片二值化处理，把[240, 240, 240]~[255, 255, 255]以外的颜色变成0
thresh = cv2.inRange(img, np.array([0, 0, 200]), np.array([70, 70, 255]))
#创建形状和尺寸的结构元素
kernel = np.ones((3, 3), np.uint8)
#扩张待修复区域
mask = cv2.dilate(thresh, kernel, iterations=1)/255.0
spectrum_show(mask)
# print(mask.max(),mask.min())
pic=gen_pic_with_mask(mask,img)
# pic= cv2.imread("cat_noise.jpg") #测试去噪
# mask = None #测试去噪
epsilon = 0.1
dt = 0.1
lambda_ = 0.1 #任意改变lambda不影响结果，可置为0
epochs=601
pic = (pic/255.0).astype(np.float)
pic_copy = np.zeros(pic.shape)
for epoch in range(epochs):
    # 每epoch次显示一次数据，保存一次数据
    if epoch % 100 ==0:
        print('epoch,当前的循环次数：',epoch, np.abs(pic- pic_copy).max())
        # spectrum_show(np.abs(pic- pic_copy))
        cv2.imwrite("dog_filed"+str(epoch)+'.jpg',img=np.uint8(pic*255))
    pic_copy = pic.copy()
    if epoch<epochs-1:
        for i in range(3):
            pic[:,:,i] = tv_inpaint(pic_copy[:,:,i],mask,epsilon,dt,lambda_)
pic=np.uint8(pic*255)
spectrum_show(pic)
```

8 图像修复

图像修补 NS、FMM

```
img= cv2.imread("dog_defiled.jpg")
# mask = (img[:,:,2]>220) & (img[:,:,0]<100)
#图片二值化处理，把[240, 240, 240]~[255, 255, 255]以外的颜色变成0
thresh = cv2.inRange(img, np.array([0, 0, 200]), np.array([70, 70, 255]))
#创建形状和尺寸的结构元素
kernel = np.ones((3, 3), np.uint8)
#扩张待修复区域
mask = cv2.dilate(thresh, kernel, iterations=1)
spectrum_show(mask)



out = cv2.inpaint(img, mask, inpaintRadius=-1, flags=cv2.INPAINT_TELEA) #cv2.INPAINT_TELEA cv2.INPAINT_NS

# out = inpaint.inpaint_biharmonic(image=img, mask=mask, multichannel=True )
# out=np.uint8(out*255)
# print(np.amax(out))
spectrum_show(out)
cv2.imwrite("dog_mask.jpg", mask)
cv2.imwrite("dog_filed.jpg", out)
```

图像修补 PatchMatch

网上能找到源码，并且代码较多，请自行搜索

◆ 接下来的时间：上机实验并完成实验报告

实验 08：图像修复

姓名		学号	
实验地点		实验日期	

一、实验内容

【1】任选图片，进行噪声模拟与统计排序滤波器的恢复实验。

【2】任选图片，用退化模型进行退化模拟，并通过维纳滤波进行复原，需测试噪声的影响。

【3】选合适的图片，进行破损图像修补的实验。