画像処理実験 第 5 回

学生番号: 09B23523
大野颯也 (OHNO, Soya)

2025 年 10 月 24 日

1　第 5 回

1.1　課題 5.2

1.1.1　極大点リストから，「特徴点らしさ」の大きいものを MAX 個選び出す

大きいものから選び出す必要があるため， ary をソートする．以下がソートに使用した関数である．

```
 1: def quicksort_des(ary,low,high):
 2:   if low>=high:
 3:     return
 4:   pivot=ary[(low+high)//2,2]
 5:   i,j=low,high
 6:   while i<=j:
 7:     while ary[i,2]>pivot:
 8:       i+=1
 9:     while ary[j,2]<pivot:
10:       j-=1
11:     if i<=j:
12:       ary[i,:],ary[j,:]=ary[j,:].copy(),ary[i,:].copy()
13:       i+=1
14:       j-=1
15:   quicksort_des(ary,low,j)
16:   quicksort_des(ary,i,high)
```

Python では値を渡すことができず，代入渡しとなってしまうため，swap 関数部分である

```
1:       ary[i,:],ary[j,:]=ary[j,:].copy(),ary[i,:].copy()
```

では .copy() を使用して，値のみを変換するようにしている．
MAX は detectFeaturePoints 関数 に引数として渡されているため，

```
1:   quicksort_des(ary,0,len(x)-1)
2:   ary=ary[:MAX]
```

と記述することで大きいものから MAX 個選ぶことができる．

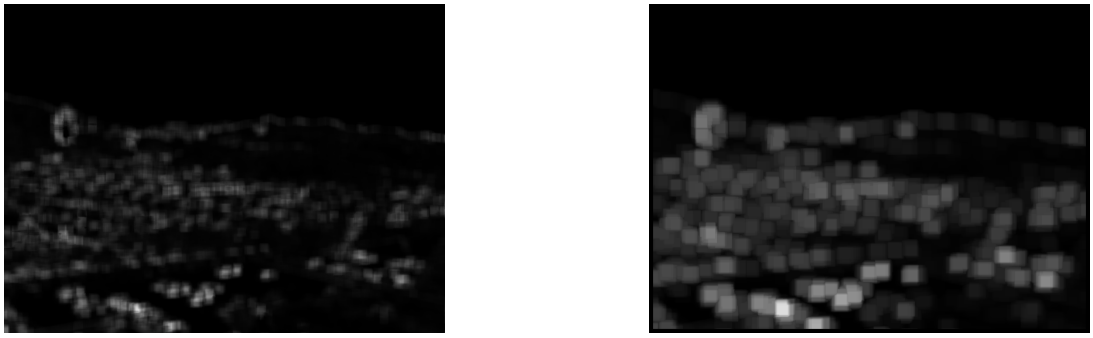
1.2　課題 5.3

1.2.1　LocalMax の説明

LocalMax 関数は次のようになっている．

```
def LocalMax(src,W=7):
# (5) 近傍での最大値検出
  out=np.zeros_like(src)
  for v in range(-W,W+1):
   for u in range(-W,W+1):
    out[W:-W,W:-W]=np.maximum(out[W:-W,W:-W],
                              src[v+W:src.shape[0]+v-W,
                                  u+W:src.shape[1]+u-W])
  return out
```

この関数は，点 [y,x] について周囲 [W:-W,W:-W] における最大値に値を置き換えるという作業を (u,v) をずらしながら行う．図 3a,3b はそれぞれ TKfilter によって絞られた特徴点の候補，LocalMax によってある特徴点候補における周辺 15×15 における最大値である．



(a) TKfilter 後の画像　　(b) LocalMax 後の画像

図 1: LocalMax による画像の変化

blur 関数では周囲 [W:-W,W:-W] の平均値を出していたので，処理は加算をするところを最大値を取得するように置き換えただけで実装できる．

1.2.2 縦横 2 段階に分解して効率化

以下は縦と横の二段階にして効率化したものである．

```
def LocalMax(src,W=7):
 out=np.zeros_like(src)
 T=np.zeros_like(src)
 for v in range(-W,W+1):
  T[W:-W,W:-W]=np.maximum(T[W:-W,W:-W],
                          src[v+W:v+src.shape[0]-W,
                              W:src.shape[1]-W])
 for u in range(-W,W+1):
  out[W:-W,W:-W]=np.maximum(out[W:-W,W:-W],
                            T[W:-W,u+W:u+src.shape[1]-W])
 return out
```

次が，(1) デフォルト (2) 効率化後 の実行速度である．

```
(1) Max:0.207 sec
(2) Max:0.036 sec
```

6 倍程度高速に動いていることがわかる．これは blur においてもそうだったが，二重ループによって 15^2 回の処理が必要だったものを，ループを 2 段階に分けることによって 15×2 回の処理で済んでいるからである．

1.2.3 FAST と NAIVE の速度・結果を比較

極大判定における，FAST な実装と NAIVE な実装について比較する．

```
(1) ary:0.002 sec
(2) ary:0.160 sec
```

(1) が FAST な実装で，(2) が NAIVE な実装である．ここまでの速度の差が出てきてしまっているのは，FAST においては行列式の計算を行っているのに対し，NAIVE においては二重ループの計算 (1024×768) を行っているからである．

FAST で行っていることは，0 の要素を省いて計算することである．厳密には，数値が 0 でない値 (0 以上) に対してその座標と値を行列として集めている．これによって高速な処理を可能にしている．以下において，np.where などの使い方を載せる．

```
  X=np.array([0,2,0,0,3])
  idx = np.where(X)[0]
  vals = X[idx]
  Mat = np.array([idx, vals])
  print(Mat)
  Mat=Mat.T
  print(Mat)

---------- 実行結果 ----------

[[1 4]
 [2 3]]
[[1 2]
 [4 3]]
```

1.2.4 全体のコード

```
# [5.1 features]

def LocalMax(src,W=7):
 out=np.zeros_like(src)
 T=np.zeros_like(src)
 for v in range(-W,W+1):
  T[W:-W,W:-W]=np.maximum(T[W:-W,W:-W],
                          src[v+W:v+src.shape[0]-W,
                              W:src.shape[1]-W])
 for u in range(-W,W+1):
  out[W:-W,W:-W]=np.maximum(out[W:-W,W:-W],
                            T[W:-W,u+W:u+src.shape[1]-W])
 return out


def detectFeaturePoints(src,MAX=100):
  W=7

  tkf=TKfilter(src)
  eb.                stopWatch('TKfilter')
  out=LocalMax(tkf)
  out[    :W+1]=0    # 周辺 8 pixel は除去
  out[-W-1:   ]=0    # ※ 1
  out[:,    :W+1]=0
  out[:,-W-1:   ]=0
  eb.                stopWatch('Max')

```

```
28: # (6) 極大判定
29: ## FAST:
30:   mask=(out==tkf)&(out>0) # (最大値 == 中心) かつ (最大値 > 0)
31:   y,x=np.where(mask)      # mask[y,x]!=0 であるような x,y のリスト
32:   ary=np.vstack((x,y,out[y,x])).T # [x,y,out[y,x]] を行とする行列
33:
34: ## NAIVE:
35: #  hgt,wdt=tkf.shape
36: #  ary=[]
37: #  for y in range(W+1,hgt-W-1):
38: #   for x in range(W+1,wdt-W-1):
39: #    if out[y,x]==tkf[y,x] and out[y,x]>0:
40: #     ary+=[[x,y,out[y,x]]]
41: #  ary=np.array(ary)
42:
43:   eb.                stopWatch('ary')
44:
45:   # (7) ary を第 3 列の降順に MAX 個残す
46:   quicksort_des(ary,0,len(ary)-1)
47:   ary=ary[:MAX]
48:   eb.                stopWatch('sort')
49:
50:   # (8) 画像で結果表示
51:   hgt,wdt=tkf.shape
52:   img=np.zeros((hgt,wdt,3),'f') # 出力画像
53:   img[:,:,0]=tkf*9 # 濃淡画像からカラー画像に変換
54:   img[:,:,1]=tkf*9 # Numpy tips: Broadcast を参照
55:   img[:,:,2]=tkf*9
56:
57:   for x,y,v in ary:
58:    x=int(x)
59:    y=int(y)
60:    img[y-W:y+W+1,x-W:x+W+1]=(img[y-W:y+W+1,x-W:x+W+1]+[1,0,0])/2
61:
62:   eb.imshow(img)
63:
64:   return ary
65:
66:
67: def main():
68:   src=eb.getImg('0.jpg') # 入力画像
69:   eb.                    stopWatch('start')
70:   ary=detectFeaturePoints(src)
71:   #print(ary)            # ary を表示
72:   eb.showTime()
73:
74:
75:
76: def quicksort_des(ary,low,high):
77:   if low>=high:
78:     return
79:   pivot=ary[(low+high)//2,2]
80:   i,j=low,high
81:   while i<=j:
82:     while ary[i,2]>pivot:
83:       i+=1
84:     while ary[j,2]<pivot:
85:       j-=1
86:     if i<=j:
87:       ary[i,:],ary[j,:]=ary[j,:].copy(),ary[i,:].copy()
88:       i+=1
89:       j-=1
90:   quicksort_des(ary,low,j)
```

```
91:    quicksort_des(ary,i,high)
```

1.2.5 実行速度

```
1: start:0.005 sec
2: TKfilter:0.036 sec
3: Max:0.037 sec
4: ary:0.002 sec
5: sort:0.005 sec
```

1.2.6 特徴点が絞られた画像

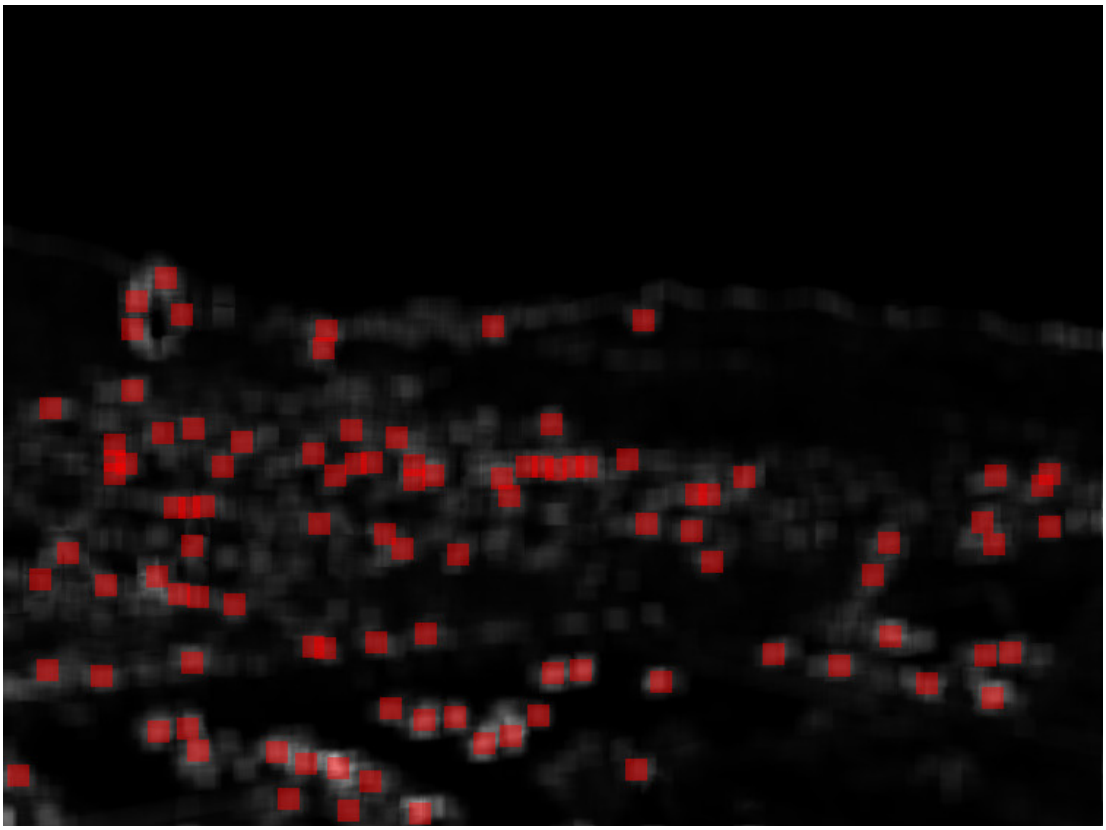
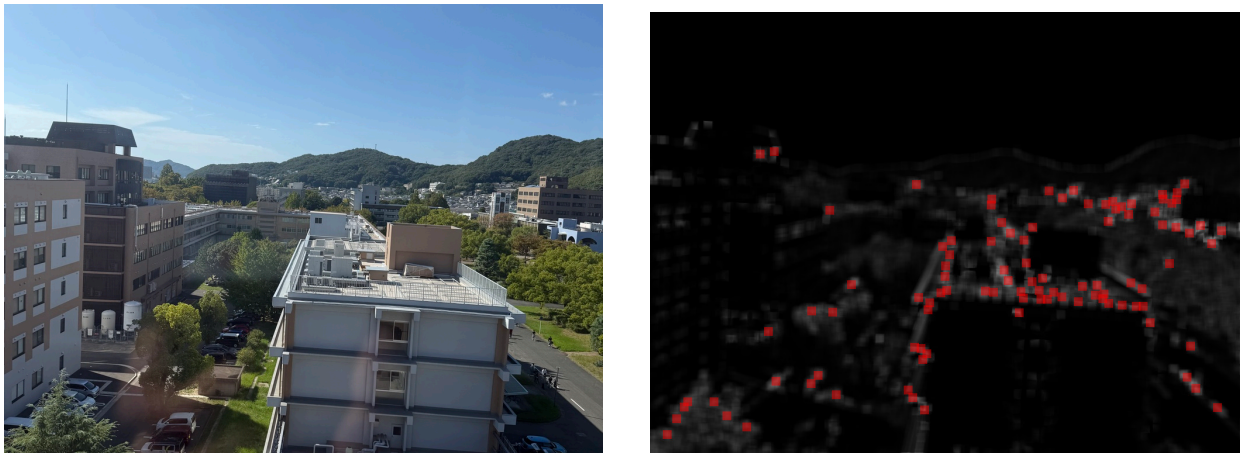


図 2: 特徴点 (周囲 15*15) が赤

1.3 自分で撮影した画像

2 感想



(a) 撮影した画像　　(b) 特徴点

図 3: 撮影した画像への実行