بهبود کنتراست تصویر

منا رداد

اطلاعات گزارش

تاریخ:
1400/9/8

واژگان کلیدی:
درونیابی
هیستوگرام
بهبود کنتراست
همسان سازی
هیستوگرام محلی

چکیده

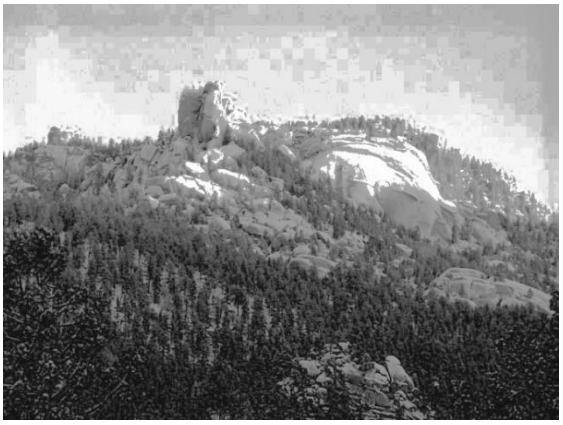
در این گزارش میخواهیم تصاویر داده شده را از لحاظ کنتراست بهبود بخشیم. یکی از روش های انجام این عملیات استفاده از روش همسان سازی هیستوگرام می باشد. در مراحل بعد از فرمول $g=\alpha.f+(1-\alpha)\ f_{he}$ استفاده کرده و با جایگزینی مقادیر متفاوت آلفا و نیز استفاده از تابع هیستوگرام تصاویر اصلی را از لحاظ کنتراست بهبود می بخشیم. از پنجره هایی با سایز های 51*51و101*101و151*151و201*201 استفاده کرده و تابع هیستوگرام داده شده را همسان سازی می کنیم و در پایان از روش هایی جهت افزایش سرعت تابع همسان سازی شده هیستوگرام استفاده می کنیم.

1-مقدمه

در این تمرین به بهبود کنتراست تصویر پرداختیم. امروزه از روش های متفاوتی جهت بهبود تصاویر استفاده می کنیم که میتوان به عنوان نمونه به کنتراست تصویر و بهبود آن اشاره کرد. در واقع این تکنیک به روش های گوناگون انجام پذیر می باشد اما یکی از این روش ها استفاده از تکنیک یکنواخت سازی هیستوگرام می باشد که کنتراست تصویر در مقایسه با تصویر اولیه بیشتر میشود که همین امر سبب افزایش دقت پردازش های بعدی و نیز بهبود کیفیت تصویر می باشد.

1-2- همسان سازی هیستوگرام

در این بخش از تمرین به کمک همسان سازی هیستوگرام میخواهیم کنتراست تصویر را بهبود دهیم. روش کار به این صورت می باشد که ابتدا پیش پردازش های لازم را انجام می دهیم و سپس مقدار gray level ها و نیز تعداد تکرار هر سطح خاکستری را محاسبه می کنیم و به صورت کلید مقدار در دیکشنری ای save می کنیم.، سپس مجموع همه این سطوح را حساب کرده و هر پیکسل را بر این مجموع تقسیم می کنیم. سپس برای هر یک از سطوح خاکستری بین 0 تا 255 فراوانی تجمعی را بدست می آوریم و مقادیر حاصل را در 255 ضرب می کنیم و پیکسل ها را با مقدار جدید جایگزین می کنیم. نتایج زیر حاصل انجام این عملیات است:



با متعادل شدن تصویر همانطور که در عکس ها مشاهده میشود جزئیات بهتر نمایان میشود و قسمت های تاریک موجود در عکس روشن تر شده و بهتر مشخص می شود.

2-2- پیاده سازی f_{he} g=α.f+(1-α)

در این بخش از تمرین فرمول f_{he} **g=α.f+(1-α)** را با دادن مقادیر مختلف به آلفا به تصویر اعمال می کنیم تا آن را بهبود بخشیم که برای این کار از یک حلقه تکرار کمک می گیریم و آلفا را از 0.1 تا 0.5 افزایش داده و در فرمول قرار می دهیم و در هر قسمت g را نیز به نمایش در می آوریم. حاصل این کار نتایج زیر می باشد :

آلفا 0.1

آلفا 0.2

آلفا 0.5

آلفا 0.3

آلفا 0.1

آلفا 0.2

آلفا 0.2

آلفا 0.3

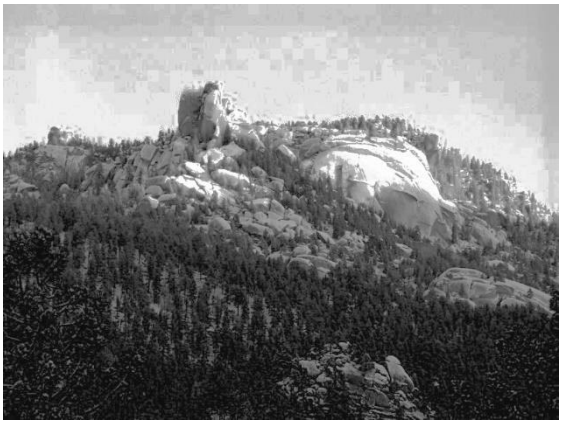
آلفا 0.4

آلفا 0.3

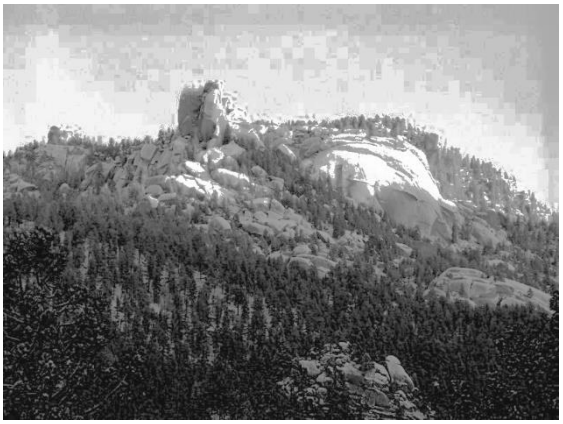
آلفا 0.4

آلفا 0.5

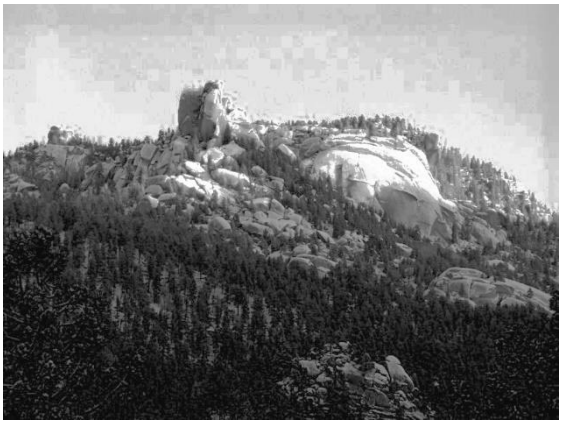
آلفا 0.5



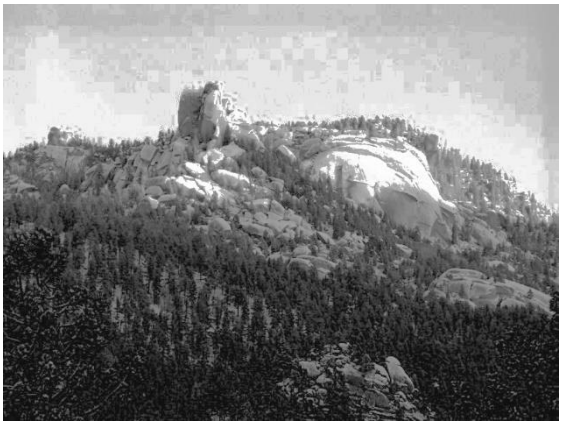
آلفا 0.3



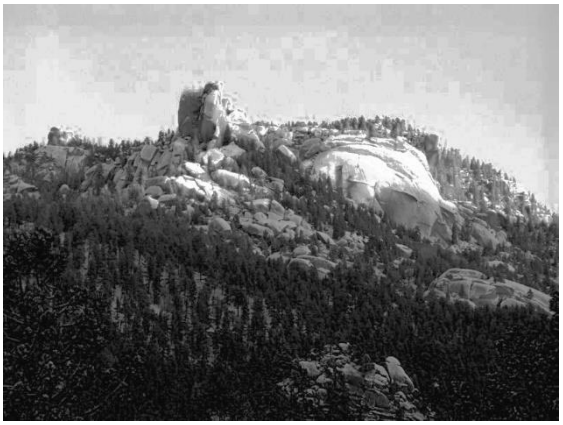
آلفا 0.1



آلفا 0.4



آلفا 0.2



آلفا 0.5

همانطور که در تصاویر بالا ملاحظه میشود هرچه ضریب عکس اولیه افزایش میابد و ضریب عکس متعادل شده کاهش پیدا می کند با کم شدن کنتراست تصویر، از جزئیات و نویز تصویر کم میشود و تصویر تاریک تر میشود..

3-2-همسان سازی محلی هیستوگرام(LHE)

در این قسمت مراحل بخش اول را دوباره اجرا می کنیم تنها با این تفاوت که از پنجره های به ابعاد 51*51و101*101و151*151و201*201 به جای یک پنجره به اندازه طول و عرض تصویر استفاده می کنیم. روش کار به این ترتیب است که در ابتدای پیکسل ها پنجره را قرار می دهیم و فقط برای همان قسمت عملیات متعادل سازی را انجام می دهیم و سپس پیکسل به پیکسل هم در طول و هم در عرض پنجره را می لغزانیم و به این ترتیب و عملیات متعادل سازی را در تصویر به صورت کامل انجام می دهیم.حال به دلیل اینکه پیکسل به پیکسل عملیات همسان سازی انجام میشود زمان اجرای آن بسیار طولانی میباشد و تصاویر پایین نتیجه این عملیات را نشان می دهد:

51*51

101*101

151*151

101*101

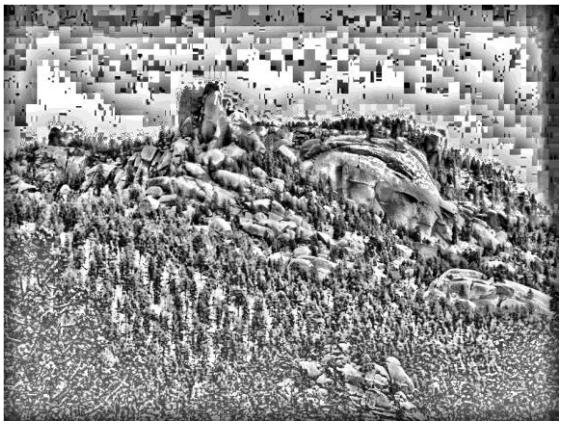
151*151

201*201

51*51

201*201

51*51



51*51

101*101

151*151

101*101

151*151

201*201

201*201

با توجه به تصاویر بالا و با بررسی آن ها متوجه می شویم که هرچه اندازه پنجره ها کمتر شود کتراست به شدت به اطلاعات محلی وابسته میشود این موضوع سبب تشخیص جزئیات در scale های پایین میشود اما سبب کاهش کیفیت تصویر هم میشود برای متوجه شدن این موضوع میتوان به تغییرات در موی افراد در تصویر he1 دقت کرد.

2-4-افزایش سرعت همسان سازی هیستوگرام محلی

در این بخش همان مراحل قسمت قبل را انجام میدهیم اما با این تفاوت که به جای لغزاندن پیکسل به پیکسل پنجره در هر مرحله پنجره را به اندازه نصف ابعاد جا به جا می کنیم.

مراجع

[1] https://stackoverflow.com/questions/50578876/histogram-equalization-using-python-and-opencv-without-using-inbuilt-functions

[2] https://medium.com/@kyawsawhtoon/a-tutorial-to-histogram-equalization-497600f270e2

2-6- کد های پیوست شده

```
he1 = cv2.imread("he1.jpg")
he2 = cv2.imread("he2.jpg")
he3 = cv2.imread("he3.jpg")
he4 = cv2.imread("he4.jpg")
def hiseq(he1):

  # preprocessing

  he1_gray = cv2.cvtColor(he1, cv2.COLOR_BGR2GRAY)
  unique, counts = np.unique(he1_gray, return_counts=True)

# count of each pixel
  dict_he1 =dict(zip(unique, counts))

# sum of all pixels
  sum_he1 = sum(dict_he1.values())

# preprocessing
  if dict_he1.get(0)==None:
    dict_he1[0]=0

# count of each pixel / sum of all pixels
  for i in dict_he1 :
      dict_he1[i] = dict_he1[i] / sum_he1

  dict_accum = {}
  dict_accum[0] = dict_he1.get(0)

# cumulative sum
  for i in range(1,256):
    if dict_he1.get(i):
      dict_accum[i] =  dict_he1.get(i) + dict_accum.get(i-1)
    else:
      dict_accum[i]= dict_accum.get(i-1)
```

```
[ ]
    # cumulative sum * max of pixels range
      for i in range(256):
          dict_accum[i] = round(dict_accum.get(i)*255)

    #  equalize image
      for i in range(he1_gray.shape[0]):
        for j in range(he1_gray.shape[1]):
           he1_gray[i][j] = dict_accum[he1_gray[i][j]]
      # cv2_imshow(he1_gray)
      return he1_gray

    hiseq(he1)
    hiseq(he2)
    hiseq(he3)
    hiseq(he4)
```

```
[ ]

    # preprocessing
    f_rgb = cv2.imread("he4.jpg")
    f = cv2.cvtColor(f_rgb,cv2.COLOR_BGR2GRAY)

    f_he = hiseq(f_rgb)

    # apply equation
    for i in np.arange(0.1,0.6,0.1):
      g = i*f+(1-i)*f_he
      cv2_imshow(g)
```

```
def LHE(he1,xWin,yWin):

  # preprocessing

  he1_gray = cv2.cvtColor(he1, cv2.COLOR_BGR2GRAY)
  he1_local= np.empty((xWin, yWin), float)

  # for m in range(0,he1_gray.shape[0]-xWin,int((xWin-1)/2)):
  #   for n in range(0,he1_gray.shape[1]-yWin,int((yWin-1)/2)):

  for m in range(he1_gray.shape[0]-xWin):
    if m+xWin > he1_gray.shape[0] :
          break
    for n in range(he1_gray.shape[1]-yWin):



        he1_local = he1_gray[m:m+xWin,n:n+yWin]

        unique, counts = np.unique(he1_local, return_counts=True)

      # count of each pixel
        dict_he1 =dict(zip(unique, counts))

      # sum of all pixels
        sum_he1 = sum(dict_he1.values())

        # preprocessing
        if dict_he1.get(0)==None:
            dict_he1[0]=0

        # count of each pixel / sum of all pixels
        for i in dict_he1 :
              dict_he1[i] = dict_he1[i] / sum_he1
```

```
        dict_accum = {}
        dict_accum[0] = dict_he1.get(0)

        # cumulative sum
        for i in range(1,256):
            if dict_he1.get(i):
              dict_accum[i] =  dict_he1.get(i) + dict_accum.get(i-1)
            else:
              dict_accum[i]= dict_accum.get(i-1)

        # cumulative sum * max of pixels range
        for i in range(256):
              dict_accum[i] = round(dict_accum.get(i)*255)

        #  equalize image
        for i in range(he1_local.shape[0]):
            for j in range(he1_local.shape[1]):
              he1_gray[i+m][j+n] = dict_accum[he1_local[i][j]]
  cv2_imshow(he1_gray)
  # return he1_gray

LHE(he1,201,201)
```