|  |
| --- |
|  |

|  |  |
| --- | --- |
|  | 🞂گزارش پروژه پردازش تصویر  Interpolation  محمد شکری: نام  دکتر صادق فدایی: استاد  981531027: شماره دانشجویی |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | هدف :  اجرای الگوریتم های nearest neighbor interpolation و bilinear interpolation بر روی یک عکس یکسان با اندازه 260 در 220 پیکسل و تبدیل آن به 512 در 512 پیکسل.  انجام MSE(Mean square error) بر روی هر دو عکس تولید شده توسط الگوریتم ها و مقایسه آن دو.  انجام :  توسط پایتون و با استفاده از ماژول PILLOW و نوشتن هر الگوریتم توسط پایتون  نتیجه گیری :  با توجه MSE، آن الگوریتم که عدد کمتری دارد بهتر است.  همچنین از لحاظ مرتبه زمانی |  |  |  |

|  |
| --- |
|  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | مقدمات :   * امروزه روش های مختلفی برای resize یک عکس وجود دارد که در این گزارش دو تای آن هارا بررسی کرده. * روش کلی کار هم بدین صورت است ک هبا استفاده از داده های معلوم داده های جدید را در مکان های جدید ، مقدارشان را تخمین زده. |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Nearest neighbor :   * در این روش برای هر ورودی داده شده ، خروجی بر اساس مقدار وابسته در dataset بدست آمده که در مقدار مستقل dataset نزدیکترین به ورودی خواهد بود. * به زبان ساده اگر عکس بزرگ شود ، هر پیکسل از عکس قبلی در عکس جدید در مختصات نسبی به اندازه نسبت آن تقریبا تکرار می شود و بلعکس. * در این روش مقدار پیکسل ها تغییری نمی کند و ما در واقع تنها جایگذاری می کنیم. |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | کد NN :   * متد nearestNeighbor از کلاس Interpolation در فایل Interpolation.py * این متد طول و عرض عکس جدید را می گیرد. * سپس scale های طولی و عرضی را به دست می آورد. * برای هر نقطه در عکس جدید ، آن را به نقطه ایی در عکس قدیم مپ می کند بدین صورت که طول و عرض فعلی را تقسیم بر scale هر کدام کرده ( باید حواسمان باشد که از طول و عرض بیرون نزنیم که با min این کار انجام می شود) |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Bilinear :   * در این روش برای هر پیکسل جدید میانگینی از وزن ها از پیکسل های پیشین به دست می آید. * در این روش مقدار پیکسل ها عوض می شود. * مختصات هر پیکسل از عکس جدید را با توجه به scale دوعکس ، نسبت می دهیم به مختصاتی در عکس قبلی و این مختصات جدید اعشاری است. * حال این مختصات جدید بین 4 پیکسل قرار می گیرد و می توانیم با floor و ceil گرفتن ار آن xMin,xMax,yMin,yMax را به دست آورده و 4 نقطه بسازیم به صورت زیر: * q12=(xMin,yMin) * q22=(xMax,yMin) * q11=(xMin,yMax) * q=(xMax,yMax) * این چهار نقطه در واقع یک مربع را تشکیل داده به ضلع 1 و نختصات ما در وسط آن قرار گرفته. * حال هر کدام از این مختصات ها ، یک مقدار پیکسل دارند و ما می خواهیم مقدار پیکسل را در مختصاتی که مد نظر است ، به دست آوریم. * نمونه :     مقدار این نقطه را اگر f(x,y) بنامیم از روش زیر به دست می آید :  Text  Description automatically generated with medium confidence |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | کد Bilinear :   * متد bilinear از کلاس Interpolation در فایل Interpolation.py * این متد طول و عرض عکس جدید را می گیرد. * سپس scale های طولی و عرضی را به دست می آورد. * برای هر نقطه در عکس جدید ، آن را به نقطه ایی در عکس قدیم مپ می کند که این کار را متد findBilinearPixel انجام می دهد. * در متد findBilinearPixel ، ابتدا مختصات فعلی را که i,j هستند را با استفاده از scale به عکس پیشین ربط می دهیم و مقدار آن در x,y قرار گرفته. * حال باید ماکزیمم و مینیمم محور x,y را حساب کنیم. * سپس 4 نقطه را تشکیل می دهیم. * ممکن است زمان هایی ما در آخرین نقطه باشیم که xMin=xMax و yMin=yMax است یا اینکه روی محور طولی حرکت کنیم که yMin=yMax یا اینکه روی محور عرضی حرکت کنیم که xMin=xMax یا اینکه چهار نقطه مجزا داشته باشیم. این 4 شرط برای این روند است. * در نهایت بر أساس فرمول ریاضی مقدار مینگین پیکسل را حساب کرده. |  |  |  |

فایل Interpolation.py

import numpy  
from PIL import Image  
  
class Interpolation:  
 def \_\_init\_\_(self, src):  
 self.img = Image.open(src)  
  
 def save(self, dst):  
 self.img.save(dst)  
 return self  
  
 def nearsetNeighbor(self, newWidth, newHeight):  
 width, height = self.img.size  
 scaleW = newWidth / (width )  
 scaleH = newHeight / (height)  
 newImg = Image.new(self.img.mode, (newWidth, newHeight), 'white')  
 for i in range(newWidth):  
 for j in range(newHeight):  
 x = min(width-1,numpy.floor(i / scaleW))  
 y = min(height-1,numpy.floor(j / scaleH))  
 pixel = self.img.getpixel((x, y))  
 newImg.putpixel((i, j), pixel)  
 self.img = newImg  
 return self  
  
 def findBilinearPixel(self, i, j, scaleW, scaleH, width, height):  
 x = i \* scaleW  
 y = j \* scaleH  
  
 xBottom = int(x)  
 yBottom = int(y)  
 xTop = min(width - 1, numpy.ceil(x))  
 yTop = min(height - 1, numpy.ceil(y))  
  
 if xTop == xBottom and yTop == yBottom:  
 return self.img.getpixel((int(x), int(y)))  
 elif xTop == xBottom:  
 q1 = self.img.getpixel((int(x), int(yBottom)))  
 q2 = self.img.getpixel((int(x), int(yTop)))  
 return q1 \* (yTop - y) + q2 \* (y - yBottom)  
 elif yTop == yBottom:  
 q1 = self.img.getpixel((int(xBottom), int(y)))  
 q2 = self.img.getpixel((int(xTop), int(y)))  
 return q1 \* (xTop - x) + q2 \* (x - xBottom)  
 p00 = self.img.getpixel((int(xBottom), int(yBottom)))  
 p01 = self.img.getpixel((int(xTop), int(yBottom)))  
 p10 = self.img.getpixel((int(xBottom), int(yTop)))  
 p11 = self.img.getpixel((int(xTop), int(yTop)))  
  
 q1 = p00 \* (xTop - x) + p01 \* (x - xBottom)  
 q2 = p10 \* (xTop - x) + p11 \* (x - xBottom)  
 return q1 \* (yTop - y) + q2 \* (y - yBottom)  
  
 def bilinear(self, newWidth, newHeight):  
 width, height = self.img.size  
 scaleW = width / newWidth  
 scaleH = height / newHeight  
 newImg = Image.new(self.img.mode, (newWidth, newHeight), 'white')  
 for i in range(newWidth):  
 for j in range(newHeight):  
 pixel = self.findBilinearPixel(i, j, scaleW, scaleH, width, height)  
 newImg.putpixel((i, j), int(pixel))  
 self.img = newImg  
 return self

فایل MSE.py

from PIL import Image  
  
class MSE:  
 def \_\_init\_\_(self, src,dst):  
 self.srcImg = Image.open(src)  
 self.dstImg = Image.open(dst)  
  
 def calculate(self):  
 width,height = self.srcImg.size  
 n = height \* width  
 summation = 0  
 for i in range(width):  
 for j in range(height):  
 summation += (self.srcImg.getpixel((i,j)) - self.dstImg.getpixel((i,j))) \*\* 2  
 print(summation,n)  
 return summation / n

فایل main.py

from Interpolation import Interpolation  
from MSE import MSE  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 Interpolation('./InputImage.bmp').nearsetNeighbor(512, 512).save('./Im\_NNI.bmp')  
 Interpolation('./InputImage.bmp').bilinear(512, 512).save('./Im\_BLI.bmp')  
  
 mseNearestNeighbor = MSE('./OriginalImage.bmp','./Im\_NNI.bmp').calculate()  
 mseBilinear = MSE('./OriginalImage.bmp','./Im\_BLI.bmp').calculate()  
 print('mse nearest neighbor =',mseNearestNeighbor)  
 print('mse bilinear =',mseBilinear)  
 if mseNearestNeighbor > mseBilinear:  
 print('bilinear is better')  
 elif mseNearestNeighbor < mseBilinear:  
 print('nearest neighbor is better')  
 else:  
 print('both are equal')

فایل Im\_BLI.bmp



فایل Im\_NNI.bmp

A person wearing a hat

Description automatically generated with medium confidence