Evaluation Lab 2

Name:- Sudarshan.M.S

RollNo :- CB.EN.U4CSE18258

Input Image :- Flower.jpg

Nearest Neighbour Interpolation:-

Code:-

```
import sys

import cv2

import numpy
```

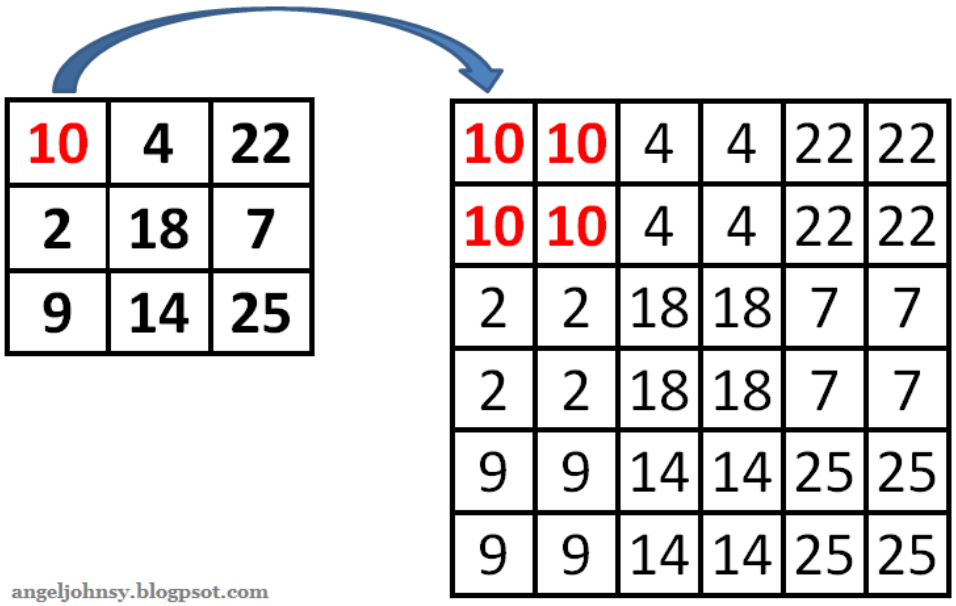
```
img_path = sys.argv[1]
img = cv2.imread(img_path)
def bound(newx, newy, width, height):
  if newx < 0:
    newx = 0
  if newy < 0:
    newy = 0
  if newx >= width:
    newx = width - 1
  if newy >= height:
    newy = height - 1
  return newx, newy


def resize_nearest(src, tx, ty):
  h, w, c = src.shape
  hratio = h/ty
  wratio = w/tx
  img = numpy.zeros((ty,tx,c), src.dtype)
  for y in range(img.shape[0]):
```

```
        for x in range(img.shape[1]):
          for c in range(img.shape[2]):
            new_x, new_y = bound(tx, ty, w, h)
            new_y = int (y * hratio + 0.5)
            new_x = int (x * wratio + 0.5)
            img[y, x, c] = src[new_y, new_x, c]
    return img


new_image = resize_nearest(img, 200, 160)
cv2.imwrite('flower_original.jpg', img)
cv2.imwrite('flower_output.jpg', new_image)
```

Output:-

Explanation:-

This method captures the intensity of nearest neighbour of the unknown pixel values and applies it to that pixel.



Bilinear Interpolation :-

Code:-

```
import sys
import cv2
import numpy
img_path = sys.argv[1]
img = cv2.imread(img_path)
def bound(newx, newy, width, height):
_if newx < 0:
___newx = 0
_if newy < 0:
___newy = 0
```

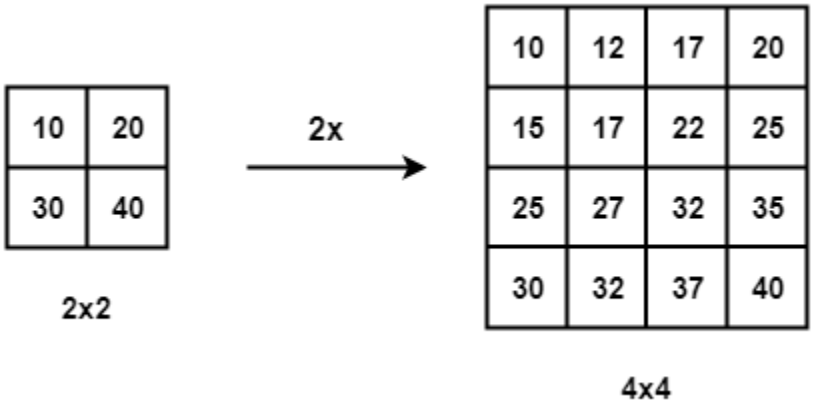
```
_if newx >= width:
___newx = width - 1
_if newy >= height:
___newy = height - 1
_return newx, newy



def resize_nearest(src, tx, ty):
_h, w, c = src.shape
_hratio = h/ty
_wratio = w/tx
_img = numpy.zeros((ty,tx,c), src.dtype)
_for y in range(img.shape[0]):
___for x in range(img.shape[1]):
_____for c in range(img.shape[2]):
_______new_x, new_y = bound(tx, ty, w, h)
_______new_y = int (y * hratio + 0.5)
_______new_x = int (x * wratio + 0.5)
_______img[y, x, c] = src[new_y, new_x, c]
_return img
```

```
new_image = resize_nearest(img, 200, 160)
cv2.imwrite('flower_original.jpg', img)
cv2.imwrite('flower_output.jpg', new_image)
```

Output:-

Explanation:-

For bilinear interpolation, the block uses the weighted average of two translated pixel values for each output pixel value.



Bicubic Interpolation

Code:-

```
import cv2
import numpy as np
import time
import math
import sys, time


def inter_kern(s,a):
    if (abs(s) >=0) & (abs(s) <=1):
        return (a+2)*(abs(s)**3)-(a+3)*(abs(s)**2)+1
    elif (abs(s) > 1) & (abs(s) <= 2):
```

```
____return a*(abs(s)**3)-(5*a)*(abs(s)**2)+(8*a)*abs(s)-4*a
__return 0



def padding(img,Height,Weight,C):
__zimg = np.zeros((Height+4,Weight+4,C))
__zimg[2:Height+2,2:Weight+2,:C] = img
__#Pad the first/last two col and row
__zimg[2:Height+2,0:2,:C]=img[:,0:1,:C]
__zimg[Height+2:Height+4,2:Weight+2,:]=img[Height-1:Height,:,:]
__zimg[2:Height+2,Weight+2:Weight+4,:]=img[:,Weight-1:Weight,:]
__zimg[0:2,2:Weight+2,:C]=img[0:1,:,:C]
__#Pad the missing eight points
__zimg[0:2,0:2,:C]=img[0,0,:C]
__zimg[Height+2:Height+4,0:2,:C]=img[Height-1,0,:C]
__zimg[Height+2:Height+4,Weight+2:Weight+4,:C]=img[Height-
1,Weight-1,:C]
__zimg[0:2,Weight+2:Weight+4,:C]=img[0,Weight-1,:C]
__return zimg


#
https://github.com/yunabe/codelab/blob/master/misc/terminal_progr
essbar/progress.py
```

```
def get_progressbar_str(progress):
__END = 200
__MAX_LEN = 50
__BAR_LEN = int(MAX_LEN * progress)
__return ('Progress:[' + '*' * BAR_LEN +
______('^' if BAR_LEN < MAX_LEN else '') +
______' ' * (MAX_LEN - BAR_LEN) +
______'] %.1f%%' % (progress * 100.))



def bicubic(img, ratio, a):

__H,W,C = img.shape

__img = padding(img,H,W,C)

__dH = math.floor(H*ratio)
__dW = math.floor(W*ratio)
__dst = np.zeros((dH, dW, 3))

__h = 1/ratio
```

```
__print('Start bicubic interpolation')
__print('It will take a little while...')
__inc = 0
__for c in range(C):
____for j in range(dH):
______for i in range(dW):
________x, y = i * h + 2 , j * h + 2

________x1 = 1 + x - math.floor(x)
________x2 = x - math.floor(x)
________x3 = math.floor(x) + 1 - x
________x4 = math.floor(x) + 2 - x

________y1 = 1 + y - math.floor(y)
________y2 = y - math.floor(y)
________y3 = math.floor(y) + 1 - y
________y4 = math.floor(y) + 2 - y

________mat_l =
np.matrix([[inter_kern(x1,a),inter_kern(x2,a),inter_kern(x3,a),inter_ker
n(x4,a)]])
________mat_m = np.matrix([[img[int(y-y1),int(x-x1),c],img[int(y-
y2),int(x-x1),c],img[int(y+y3),int(x-x1),c],img[int(y+y4),int(x-x1),c]],
```

```
                    [img[int(y-y1),int(x-x2),c],img[int(y-y2),int(x-x2),c],img[int(y+y3),int(x-x2),c],img[int(y+y4),int(x-x2),c]],
                    [img[int(y-y1),int(x+x3),c],img[int(y-y2),int(x+x3),c],img[int(y+y3),int(x+x3),c],img[int(y+y4),int(x+x3),c]],
                    [img[int(y-y1),int(x+x4),c],img[int(y-y2),int(x+x4),c],img[int(y+y3),int(x+x4),c],img[int(y+y4),int(x+x4),c]]])
        mat_r = np.matrix([[inter_kern(y1,a)],[inter_kern(y2,a)],[inter_kern(y3,a)],[inter_kern(y4,a)]])
        dst[j, i, c] = np.dot(np.dot(mat_l, mat_m),mat_r)


        # Print progress
        inc = inc + 1
        sys.stderr.write('\r\033[K' + get_progressbar_str(inc/(C*dH*dW)))
        sys.stderr.flush()
    sys.stderr.write('\n')
    sys.stderr.flush()
    return dst

path = sys.argv[1]
img = cv2.imread(path)
```

```
ratio = 3

a = -1/3

start_time = time.time()

dst = bicubic(img, ratio, a)

print("Time Elapsed"+str(time.time()-start_time))
print("___COMPLETED!!__")
cv2.imwrite('bicubic_flower.jpg', dst)
```

Output:-

Explanation:-

Like **Bilinear**, it looks at surrounding pixels, but the equation it uses is much more complex and calculation intensive, producing smoother tonal gradations.