**NUST**

**SEECS**

**Department of Computing**

**EE433: Digital Image Processing**

**Lab 2**: Image up-scaling/down-scaling from scratch

**Date:** 27th January 2020

**Instructor:** Dr. Asif Ali

**Submitted by:**

**Name:** M. Hasnain Naeem

**Reg #:** 212728

**Class:** BSCS 7-B

# Lab 2 – Downscaling & Upscaling from Scratch

## Instructions

Task 1:

* Read “lena.tiff” from directory.
* Write a program which traverses each pixel of image and perform down-scaling as follows:
  + The ratio for down-scaling should be 50%.
  + Use averaging of cell pixels to approximate better output image.
  + Use *select-one* approach to down-scale.
  + Save both images.
  + Compare the two images and express your findings.
* Repeat all of the above for 25% down-scaling
* Use opencv’s **resize** function and admire the workings of cv2.resize function especially with interpolation techniques presented in Figure 3.

Task 2:

* Read “lena.tiff” from directory.
* Write a program which traverses each pixel of the input image and perform up-scaling as follows:
  + The ratio for up-scaling should be 200% (i.e. cell size of 2x2)
  + Use interpolation as discussed in theory part and calculate unknown pixel values.
  + Save the output image.
* Express your findings on how interpolation up-scaled an image.
* Use opencv’s **resize** function and admire the workings of cv2.resize function especially with interpolation techniques presented in Figure 3. **Submission Guidelines:-**

### Task 1:

#### Findings

Pixels within image are decreased as expected.

* Average downsampling gave smoother & blurry image.
* Select one downsampling gave more clear but noisy image.

### Task 2:

#### Findings

* New size of the image is 200%.
  + But image became blurry.
  + Because, edges are not approximated well in upscaling when average is used for calculation of missing pixels.
* Image upscaled using OpenCV is smoother as well as blurry as compared to the image upscaled manually.

**Note:**

1. Task 1: General solution is provided. Besides only (2, 2) and (4,4), any filter size can be used for downsampling.
2. Task 2 code doesn’t support any filter size. But, efforts were made to incorporate the changes required for general solution.

## Code

|  |
| --- |
| *import* cv2 *from* PIL *import* Image *import* os *import* numpy *as* np   *def* show\_image(img):  *# show image using OpenCV; extra lines are for jupyter notebook* cv2.imshow("Gray Image", img)  cv2.waitKey(0)  cv2.destroyAllWindows()   *def* pixel\_average(cell\_pixels, img\_filter):  *"""  cell\_pixels: list of cell pixels  img\_filter: tuple containing filter dimensions  Returns average of the cell pixels   """* sum\_of\_pixels = np.sum(cell\_pixels)  *return* sum\_of\_pixels // (img\_filter[0] \* img\_filter[1]) *# return average   def* select\_one(cell\_pixels, img\_filter=*None*):  *"""  cell\_pixels: list of cell pixels  img\_filter: tuple containing filter dimensions  Returns first pixel from the cell list   """  # add more functionality for select one filter here, if needed.  return* cell\_pixels[0, 0]   *# complexity: No. of pixels in image (N\*M) def* downscale(img, img\_filter=(2, 2), filter\_func=pixel\_average):  *"""  img: numpy array of image to be downscaled  img\_filter: filter dimensions. Default value downscales by 50%  filter\_func: function to handle the cell pixels  """   # iterate image to downscale* downscaled\_img\_x\_dim = img.shape[0] // img\_filter[0]  downscaled\_img\_y\_dim = img.shape[1] // img\_filter[1]   *# empty 2-D array for image* new\_image = np.zeros((downscaled\_img\_x\_dim, downscaled\_img\_y\_dim))   *# to iterate new image pixels* i\_new = 0  j\_new = 0   *# Outer two loops have complexity: (image\_size/(filter\_size for rows) \* image\_size/(filter\_size for columns))  # in our case: 256\*256  # Including the filter complexity (2\*2), overall complexity for downsampling becomes:  # 256\*256\*2\*2 = 512 \* 512 = No. of pixels in image  for* i *in* range(0, img.shape[0], img\_filter[0]): *# rows* j\_new = 0  *for* j *in* range(0, img.shape[1], img\_filter[1]): *# columns* cell\_pixels = img[i:i + img\_filter[0], j:j + img\_filter[1]]  new\_image[i\_new][j\_new] = filter\_func(cell\_pixels, img\_filter)  j\_new += 1  i\_new += 1  *return* new\_image   *def* bylinear\_interpolation(cell\_pixels, img\_filter):  *# copy dimensions* curr\_dims = (cell\_pixels.shape[0], cell\_pixels.shape[1])  new\_dims = (cell\_pixels.shape[0] \* img\_filter[0], cell\_pixels.shape[1] \* img\_filter[1])   *# initialize the new image* new\_cell = np.zeros(new\_dims)   *# copy the corner values of current cell to new cell* new\_cell[0][0] = cell\_pixels[0][0]  new\_cell[0][new\_dims[1] - 1] = cell\_pixels[0][curr\_dims[1] - 1]  new\_cell[new\_dims[0] - 1][0] = cell\_pixels[curr\_dims[0] - 1][0]  new\_cell[new\_dims[0] - 1][new\_dims[1] - 1] = cell\_pixels[curr\_dims[0] - 1][curr\_dims[1] - 1]   *# approximate the values for the outer rows  for* i *in* [0, 3]:  *for* j *in* [1, 2]:  *if* i == 0:  *if* j == 1:  new\_cell[i][j] = int(3 / 4 \* new\_cell[0][0] + 1 / 4 \* new\_cell[0][3])  *elif* j == 2:  new\_cell[i][j] = int(1 / 4 \* new\_cell[0][0] + 3 / 4 \* new\_cell[0][3])  *elif* i == 3:  *if* j == 1:  new\_cell[i][j] = int(3 / 4 \* new\_cell[3][0] + 1 / 4 \* new\_cell[3][3])  *elif* j == 2:  new\_cell[i][j] = int(1 / 4 \* new\_cell[3][0] + 3 / 4 \* new\_cell[3][3])   *# approximate the values for the outer columns  for* i *in* [1, 2]:  *for* j *in* [0, 3]:  *if* i == 1:  *if* j == 0:  new\_cell[i][j] = int(3 / 4 \* new\_cell[0][0] + 1 / 4 \* new\_cell[3][0])  *elif* j == 3:  new\_cell[i][j] = int(1 / 4 \* new\_cell[0][0] + 3 / 4 \* new\_cell[3][0])  *elif* i == 2:  *if* j == 0:  new\_cell[i][j] = int(3 / 4 \* new\_cell[0][3] + 1 / 4 \* new\_cell[3][3])  *elif* j == 3:  new\_cell[i][j] = int(1 / 4 \* new\_cell[0][3] + 3 / 4 \* new\_cell[3][3])   *# approximate the values for the inner pixels  for* i *in* [1, 2]: *# because last row is already complete  for* j *in* [1, 2]:  *if* i == 1:  new\_cell[i][j] = int(3 / 4 \* new\_cell[0][j] + 1 / 4 \* new\_cell[3][j])  *elif* i == 2:  new\_cell[i][j] = int(1 / 4 \* new\_cell[0][j] + 3 / 4 \* new\_cell[3][j])  *return* new\_cell   *def* upscale(img, img\_filter=(2, 2), filter\_func=bylinear\_interpolation):  *"""  img: numpy array of image to be upscaled  img\_filter: filter dimensions. Default value upscales to 200%  filter\_func: function to handle the cell pixels  """   # iterate image to downsample* upscaled\_img\_x\_dim = img.shape[0] \* img\_filter[0]  upscaled\_img\_y\_dim = img.shape[1] \* img\_filter[1]   *# empty 2-D array for image* new\_image = np.zeros((upscaled\_img\_x\_dim, upscaled\_img\_y\_dim))   *# iterating through the current image  for* i *in* range(0, img.shape[0], img\_filter[0]): *# rows  for* j *in* range(0, img.shape[1], img\_filter[1]): *# columns   # select a 2\*2 cell* cell\_pixels = img[i:i + img\_filter[0], j:j + img\_filter[1]]   *# upscale the cell* new\_cell = filter\_func(cell\_pixels, img\_filter)   *# copy the new cell to new image  for* k *in* range(i \* 2, (i \* 2) + new\_cell.shape[0]):  *for* l *in* range(j \* 2, (j \* 2) + new\_cell.shape[1]):  new\_image[k][l] = new\_cell[k - (i \* 2)][l - (j \* 2)]   *return* new\_image   *def* \_\_main\_\_():  print()  *# open and convert file to JPEG* file\_dir = "files/"  original\_file = os.path.join(file\_dir, "lena.tiff")  img = Image.open(original\_file)  *# save* img.save("files/lena.jpeg", "JPEG", quality=100)   *# open file and convert to grey scale* filename = os.path.join(file\_dir, "lena.jpeg")  img = cv2.imread(filename)  gray\_img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)   print("Shape of image:")  print(gray\_img.shape)   """  Task 1 - Downscaling  """  print("\n\t\t\*\*\*\*\* Task 1 \*\*\*\*\*\*\*\n")  """  Part 1 - 50% Downscaling  """  *# filter to downscale* filter\_dims = (2, 2)   *# Average filtering* print("Downscaling to 50%:")  downscaled\_img = downscale(gray\_img, filter\_dims, pixel\_average)  print("Shape of downscaled image using average filtering:")  print(downscaled\_img.shape)  cv2.imwrite('Files/50%\_downscaled\_using\_average\_filter.jpg', downscaled\_img)  print("Downscaled image saved.")  print()   *# Select one filtering* downscaled\_img = downscale(gray\_img, filter\_dims, select\_one)  print("Shape of downscaled image using select one filtering:")  print(downscaled\_img.shape)  cv2.imwrite('Files/50%\_downscaled\_using\_select\_one\_filter.jpg', downscaled\_img)  print("Downscaled image saved.")  print()  print()   """  Part 2 - 25% Downscaling  """   *# filter to downscale* filter\_dims = (4, 4)   *# Average filtering* print("Downscaling to 25%")  downscaled\_img = downscale(gray\_img, filter\_dims, pixel\_average)  print("Shape of downscaled image using average filtering:")  print(downscaled\_img.shape)  cv2.imwrite('Files/25%\_downscale\_using\_average\_filter.jpg', downscaled\_img)  print("Downscaled image saved.")  print()   *# Select one filtering* downscaled\_img = downscale(gray\_img, filter\_dims, select\_one)  print("Shape of downscaled image using select one filtering:")  print(downscaled\_img.shape)  cv2.imwrite('Files/25%\_downscaled\_using\_select\_one\_filter.jpg', downscaled\_img)  print("Downscaled image saved.")  print()  print()   """  Part 3 - Downscaling using OpenCV  """  print("Downscaling to 50% using OpenCV:")  new\_dims = (gray\_img.shape[0] // 2, gray\_img.shape[1] // 2)  downscaled\_img = cv2.resize(gray\_img, new\_dims)  print("Shape of downscaled image using OpenCV:")  print(downscaled\_img.shape)  cv2.imwrite('Files/50%\_downscaled\_using\_opencv.jpg', downscaled\_img)  print("Downscaled image saved.")  print()  print()   """  Task 2  """  print("\n\t\t\*\*\*\*\* Task 2 \*\*\*\*\*\*\*\n")  *# open and convert file to JPEG* file\_dir = "files/"  original\_file = os.path.join(file\_dir, "lena.tiff")  img = Image.open(original\_file)  *# save* img.save("files/lena.jpeg", "JPEG", quality=100)   *# open file and convert to grey scale* filename = os.path.join(file\_dir, "lena.jpeg")  img = cv2.imread(filename)  gray\_img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)   print("Shape of image:")  print(gray\_img.shape)   """  Part 1 - Upscale to 200% using Bilinear Interpolation  """  print("Upscaling to 200%:")  *# filter to upscale* up\_filter\_dims = (2, 2)   *# Average filtering* upscaled\_img = upscale(gray\_img, up\_filter\_dims, bylinear\_interpolation)  print("Shape of upscaled image using average filtering:")  print(upscaled\_img.shape)  cv2.imwrite('Files/200%\_upscaled\_using\_average\_filter.jpg', upscaled\_img)  print("Upscaled image saved.")  print()  print()   """  Part 2 - Upscale using OpenCV  """  print("Upscaling to 200% using OpenCv:")  new\_dims = (gray\_img.shape[0] \* 2, gray\_img.shape[1] \* 2)  upscaled\_img = cv2.resize(gray\_img, new\_dims)  print("Shape of upscaled image using OpenCV:")  print(upscaled\_img.shape)  cv2.imwrite('Files/200%\_upscaled\_using\_opencv.jpg', upscaled\_img)  print("Upscaled image saved.")  print()  print()   *# main* \_\_main\_\_() |

## Output Screenshot





