SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF MICROELECTRONIC

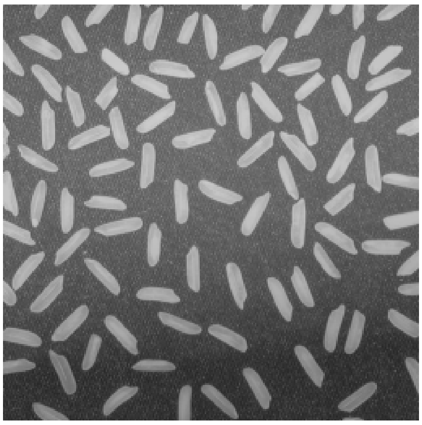
Digital Image Processing LAB Report #1

Student name: *YUAN Tong*

Course: *LAB Session I* – Professor: *YU Yajun*
Due date: *March 28th, 2025*

Question 1, Part I

Use nearest neighbor interpolation to interpolate a grey scale image shown below.



Analysis. When we apply nearest neighbor interpolation, we will first map the position of the pixels in resized image to the origin image as shown in Figure 1, then we find the nearest pixel in the origin image be python function `round()`, which is used to find the nearest integer of a float, finally we use the value of the nearest pixel to replace the value in pixel of new image.

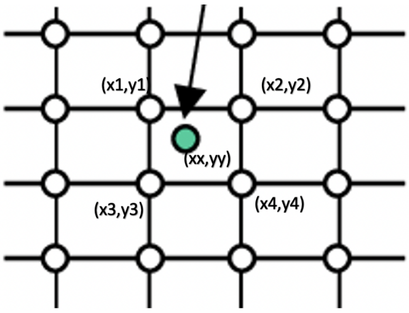


Figure 1: The position of the pixel form new image in origin image after position mapping

Algorithm 1: Nearest Neighbor Interpolation

```
input  : The origin image
output: The resized image
initialization;
special treatment for pixels at the edges;
foreach pixel x position in new image do
    foreach pixel y position in new image do
        xx ← $x \times (\frac{\text{width of source image}}{\text{width of new image}})$
        yy ← $y \times (\frac{\text{height of source image}}{\text{height of new image}})$
        output[x, y] = input[round(xx), round(yy)]
    end
end
```

```python
# Perform transform
for col in range(out_width):
    for row in range(out_height):
        x = col*((in_width-1)/(out_width-1))
        y = row*((in_height-1)/(out_height-1))
        out_image[col, row] = in_image[round(x), round(y)]
```

Question 1, Part II

Use bilinear interpolation to interpolate a grey scale image, the figure is shown in Question 1.

Analysis. When we apply bilinear interpolation, we will first find four point around the output pixels after mapping, as shown in Figure 1, the notated four points is the four points around, and we first connect each two points with same x value, then, we find the z value of the point with y position yy on two lines, finally, we connect two new points and find the z value at point (xx, yy).

Algorithm 2: Bilinear Interpolation

```
input  : The origin image
output: The resized image
initialization;
special treatment for pixels at the edges and on line;
foreach pixel x position in new image do
    foreach pixel y position in new image do
        xx ← $x \times (\frac{\text{width of source image}}{\text{width of new image}})$
        yy ← $y \times (\frac{\text{height of source image}}{\text{height of new image}})$
        (x1, y1) ← $(\texttt{round}(xx), \texttt{round}(yy))$
        (x2, y2) ← $(x1 + 1, y1)$
        (x3, y3) ← $(x1, y1 + 1)$
        (x4, y4) ← $(x1 + 1, y1 + 1)$
        left = input[x1,y1]+(input[x3,y3]-input[x1,y1])$\times(yy - y3)$
        right = input[x2,y2]+(input[x4,y4]-input[x2,y2])$\times(yy - y4)$
        output[x, y] = right + (left - right) $\times(xx - x1)$
    end
end
```

```python
def linear(x, y1, y2):
    if y2 > y1:
        return y1 + x * (y2-y1)
    else:
        return y2 + (1-x)*(y1-y2)

for col in range(out_width):
    for row in range(out_height):
        x = small_map(col*((in_width-1)/(out_width-1)), out_width)
        y = small_map(row*((in_height-1)/(out_height-1)), out_height)
        left=linear(y-math.floor(y), in_image[math.floor(x), math.floor(y)], in_image[math.floor(x), math.floor(y)+1])
        right=linear(y-math.floor(y), in_image[math.floor(x)+1, math.floor(y)], in_image[math.floor(x)+1, math.floor(y)+1])

        out_image[col, row] = round(linear(x-math.floor(x), left, right))

print(out_image)
```

Question 2

Use Python function "interp2d" from packet "scipy" or your own written algorithm to interpolate a grey scale image by using bicubic interpolation.

Analysis. When we applying bicubic interpolation to resize a image, we first choose a 4×4 block with the output pixel at the middle, instead of using function `interp1d`, we use cubic function `interp1d` to fit the every line, later, we pick the point on the lines with the same y as yy and connect four point by a smooth line, finally we pick the point (xx, yy) on the smooth line.

Algorithm 3: Bilinear Interpolation

```
input  : The origin image
output: The resized image
initialization;
special treatment for pixels at the edges and on line;
foreach pixel x position in new image do
    foreach pixel y position in new image do
        pix_x ← $x \times (\frac{\text{width of source image}}{\text{width of new image}})$
        pix_y ← $y \times (\frac{\text{height of source image}}{\text{height of new image}})$
        line = [0, 0, 0, 0];
        for $i < 4$ do
            line[i]=interp1d(xs,ys,cubic)(pix_y);
        end
        out_image[x, y]=interp1d(xs, line)(pix_y)
    end
end
```

```python
def small_map2(x, range):
    ratio = (x-1)/(range)
    return 2.01 + ratio*(range-4.02)

# The function is used to find the y value of point with in a line
 ↪  fitted by four points
def find_value(x, y, pix_x):
    f = interpolate.interp1d(x, y, kind='cubic')
    return f(pix_x)

def bicubic_11810818(input_file, dim):
    # Load image
    in_image = io.imread(input_file)
    out_width = dim[0]
    out_height = dim[1]
    in_width = in_image.shape[0]
    in_height = in_image.shape[1]
    out_image = np.zeros(dim, dtype=np.uint8)

    # Perform Exchange
    for col in range(out_width):
        for row in range(out_height):
            pix_x = small_map2(col*((in_width-1)/(out_width-1)),
             ↪  out_width)
            pix_y = small_map2(row*((in_height-1)/(out_height-1)),
             ↪  out_height)
            line = np.zeros(4)
            for i in range(4):
                line[i] = find_value(
                    [math.floor(pix_y) - 1, math.floor(pix_y),
                     ↪  math.floor(pix_y) + 1, math.floor(pix_y) + 2],
                    in_image[math.floor(pix_x)-1+i,
                     ↪  math.floor(pix_y)-1:math.floor(pix_y)+3],
                    pix_y
                )

            out_image[col, row] = find_value(
                [math.floor(pix_x)-1, math.floor(pix_x),
                 ↪  math.floor(pix_x)+1, math.floor(pix_x)+2],
                line,
                pix_x
            )

    # Save Image
    io.imsave("shrank_bicubic_11810818.tif", out_image)
```

Output images and comparison

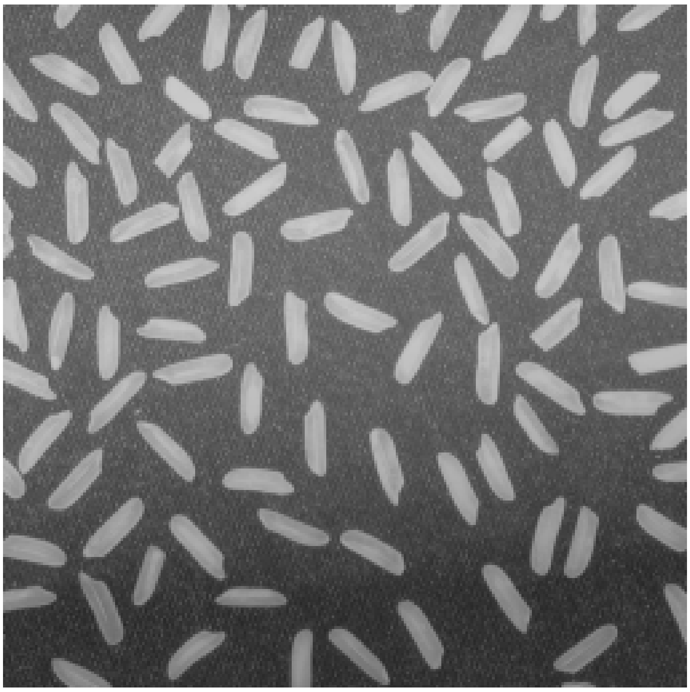


Figure 2: The original image

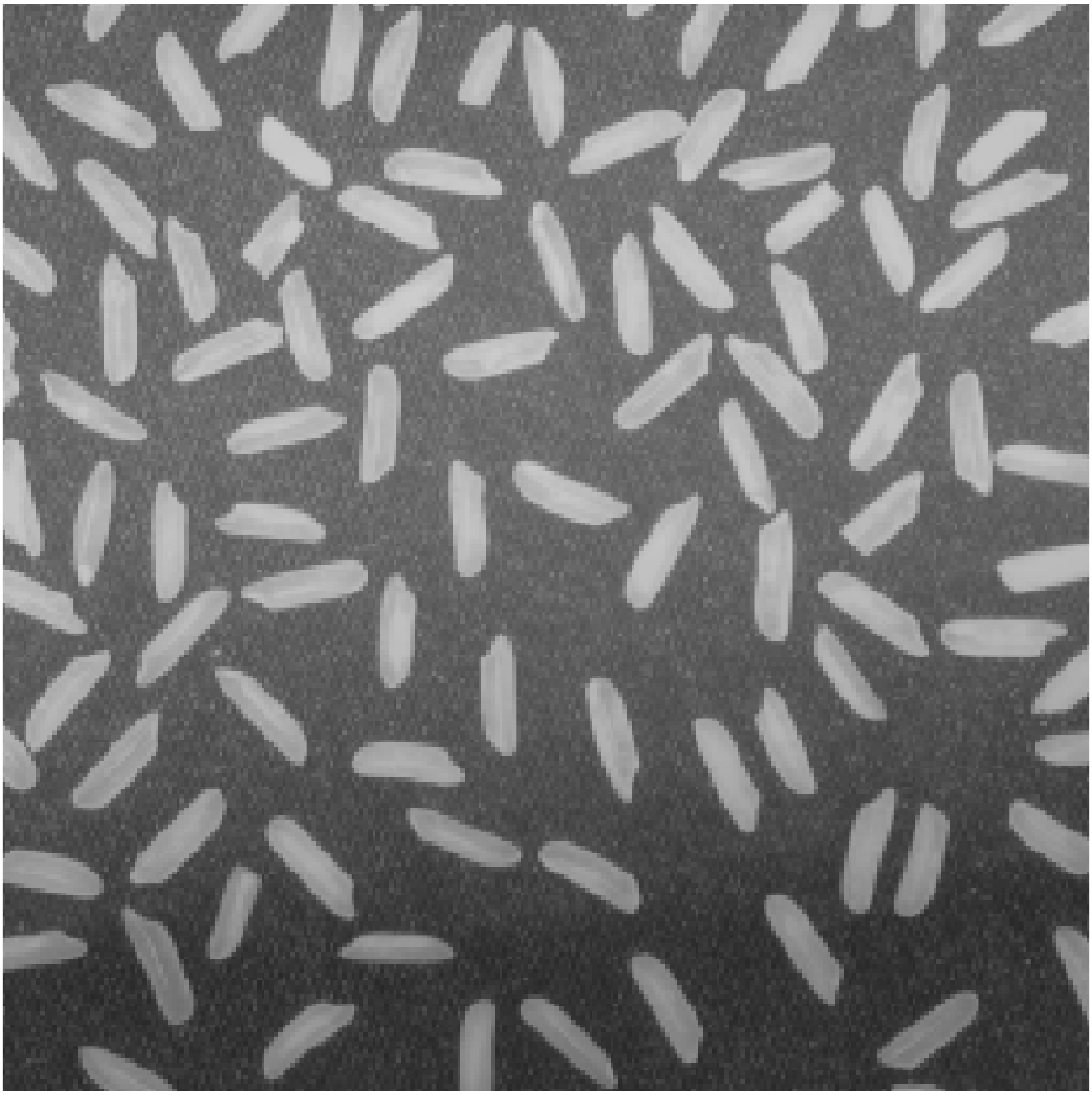


Figure 3: enlarged image with nearest neibor interpolation

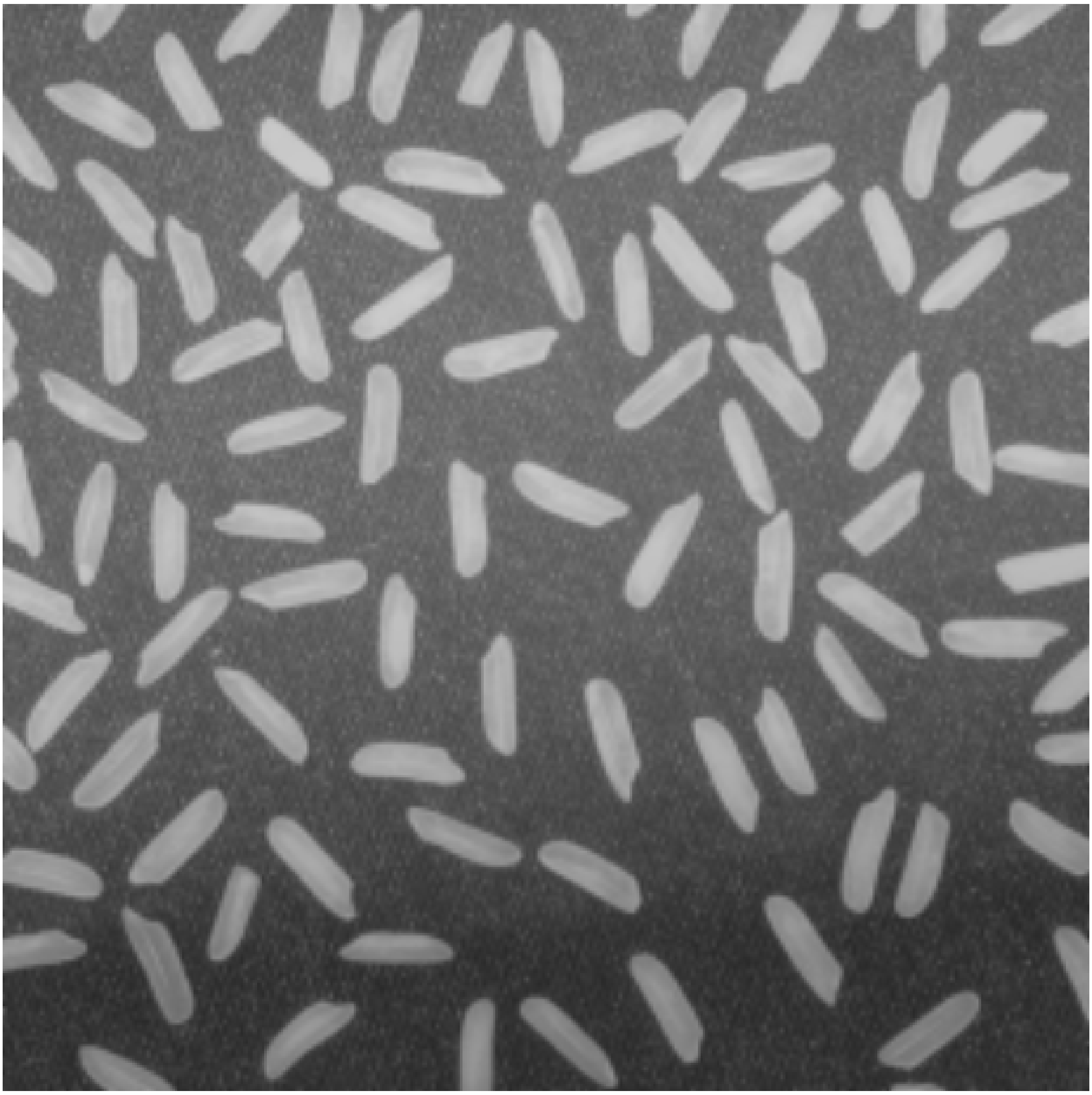


Figure 4: enlarged image with bilinear interpolation

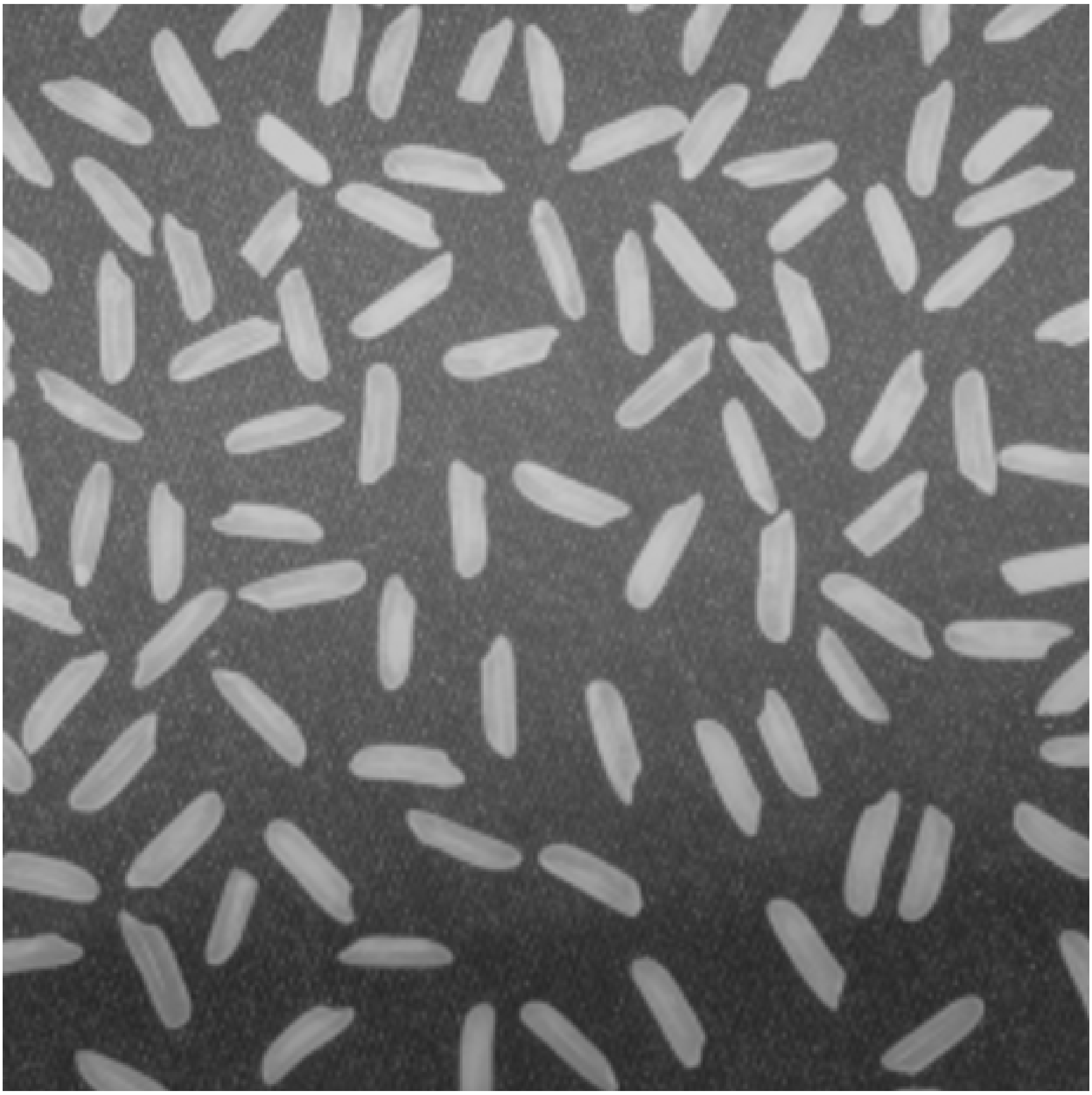


Figure 5: enlarged image with bicubic interpolation

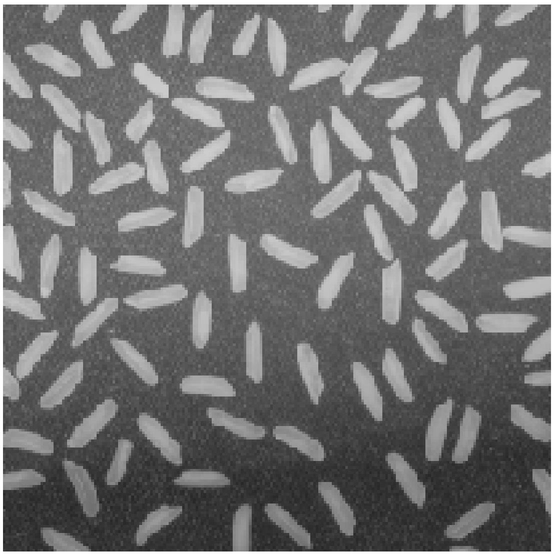


Figure 6: shrank image with nearest neibor interpolation

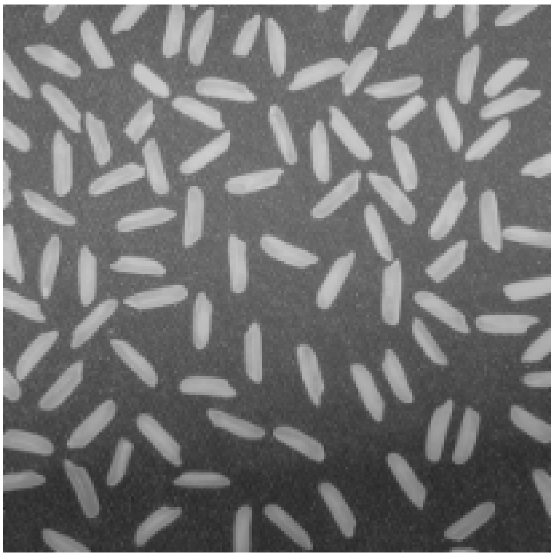


Figure 7: shrank image with bilinear interpolation

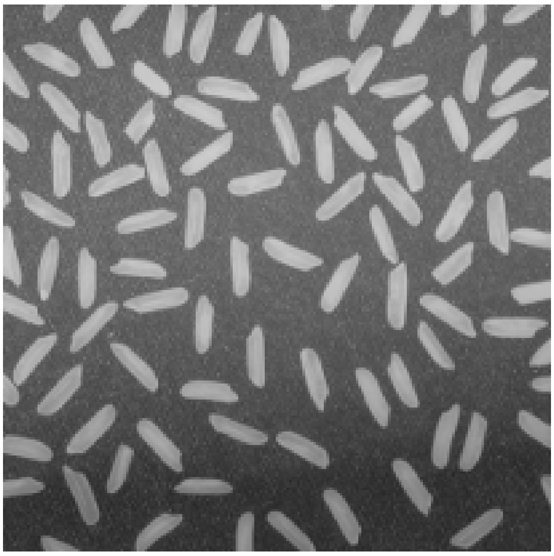


Figure 8: shrank image with bicubic interpolation

Analysis. By cooperation, we find that either for enlarged or for shrank image the bilinear performs better than nearest neighbor and bicubic performs better than bilinear.