Question 1

(a) Histogram Computation

An image histogram is a representation of occurence of each grayscale value. This can be developed to show the distribution of RGB values of color images as well. The histogram of image 1a is calculated using opencv's cv2.calHist() method.

```
hist_emma = cv.calcHist([emma_g],[0], None, [256], [0,256])
```

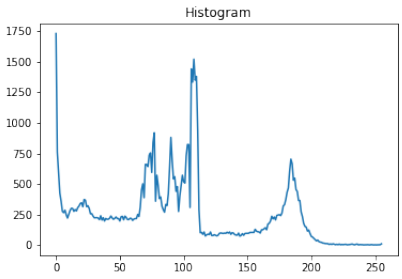
(b) Histogram Equalization

Histogram equalization distributes the intensity of an image all over the grayscale range and results which results in a flat histogram. Here the operation is carried out using cv2.equalizeHist() method. The resulting image and histogram is presented below.

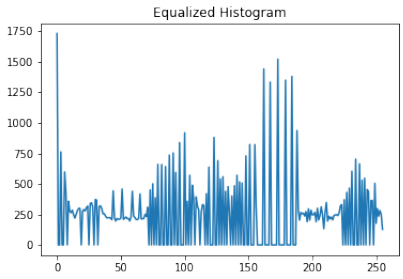
```
eq_emma = cv.equalizeHist(emma_g)
```

(a) Original grayscale image

(b) Histogram Equalized Image



(c) Histogram of 1a



(d) Equalized histogram(of 1b)

Figure 1: Histogram Calculation and Equalization

(c) Intensity Transformation

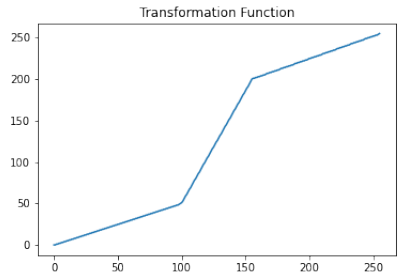
Intensity transformation has been carried out here using the transformation function in figure cv2.LUT() method is used to carry out the transformation.

```
p, q= 50,200
tr1 = np.linspace(0,p,100, dtype=np.uint8)
tr2 = np.linspace(p+1,q,56, dtype=np.uint8)
```

```
tr3 = np.linspace(q+1,255,100, dtype=np.uint8)
transform = np.concatenate((np.concatenate((tr1, tr2),
            axis = 0), tr3), axis = 0)
emma_transformed = cv.LUT(emma_c, transform)
```

(a) Original Color Image

(b) Intensity Transformed Image



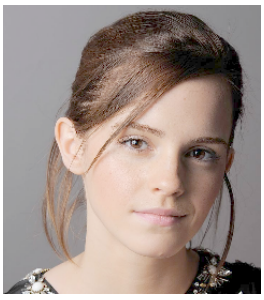
(c) Intensity transformation function

Figure 2: Intensity Transformations

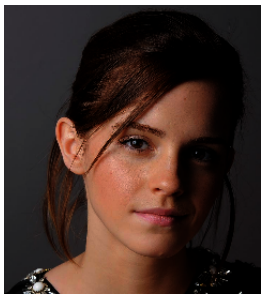
(d) Gamma Correction

Gamma correction is a non linear operation controls the luminance - brightness of a given function.

```
def gamma_correct(img, gamma):
   table = np.array([(x/255.0)**gamma*255.0 for x in np.arange(0,256)],
                                dtype = np.uint8)
   gamma_img = cv.LUT(emma_c, table)
   return gamma_img
```



(a) Gamma corrected image, gamma = 0.5



(b) Gamma corrected image, gamma = 2

Figure 3: Gamma correction

(e) Gaussian Smoothing

This is a convolution operation used to remove noise and detail.

```
k_size = 7
sigma = 1.5
kernel = cv.getGaussianKernel(k_size, sigma)
gauss_emma = cv.filter2D(emma_g, -1, kernel)
```

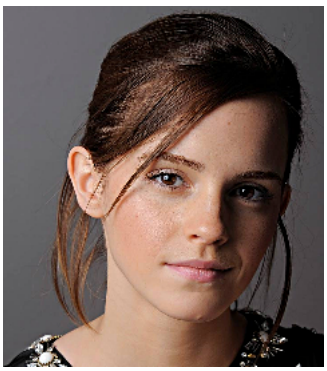
Figure 4: Gaussian Smoothing with kernal size of 7 and sigma 1.5

(f) Unsharp Masking

This is used to sharpen images by amplifying high frequency components of the image. Unsharp mask is generated by a blurred version of the original image and then it is combined with the original image to generate a sharper but less accurate version of the image.

```
gaussian = cv.GaussianBlur(emma_c, (0, 0), 2.0)
unsharp_image = cv.addWeighted(emma_c, 1.5, gaussian, -0.5, 0, emma_c)
```

(a) Blurred Image



(b) Unsharp masked image

Figure 5: Unsharp Masking using opencv

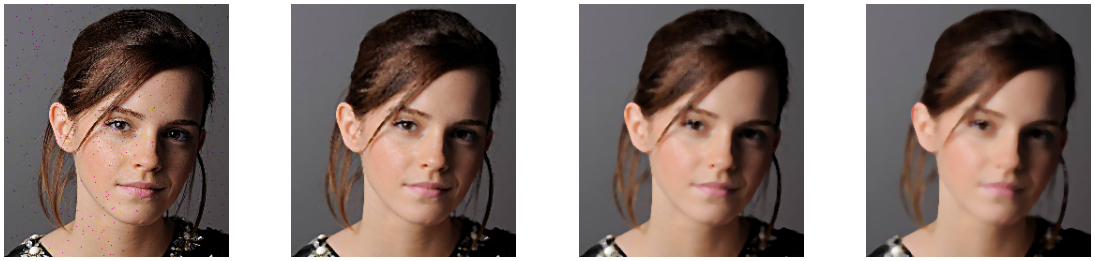
(g) Median Filtering

Median filtering is used to remove noise in images. In edge detection this is used to remove unwanted detections while preserving edges. Results are presented in figure 6

```
emma_sp = noisy(emma_c) #adding noise to image
emma_med = cv.medianBlur(emma_sp,3)
```

Question 2

Connected component analysis tool in opencv requires the image to in black and white. Therefore this counting process starts with converting the grayscale image to black

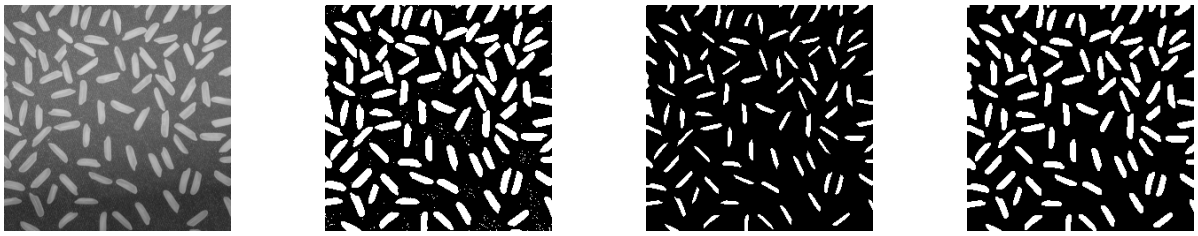


(a) Image with added noise **(b)** Median filtering, k=3 **(c)** Median filtering, k=5 **(d)** Median filtering, k=7

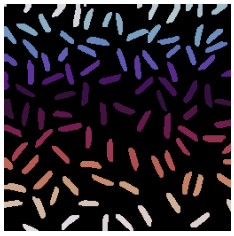
Figure 6: Median filtering

and white using thresholding. Here opencv's cv2.adaptiveThreshold() method is used to threshold the image. This increses effiency by varying threshold value in different parts of the image rather than having a single threshold value for entire image. Next cv2.erode() method is used to shrink the size of the rice grains, which removes unwanted white pixels. cv2.dilate() is used to bring back the rice grains to original size. Finally cv2.connectedComponents() is used for connected component analysis.

```
rice_thres = cv.adaptiveThreshold(rice_original, 255.0,
            cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY, 25, -10.0)
rice_eroted = cv.erode(rice_thres, np.ones((2,2)), iterations = 3)
rice_dilated = cv.dilate(rice_eroted, np.ones((2,2)), iterations = 2)
num_labels, rice_out = cv.connectedComponents(rice_dilated)
print(num_labels)
```



(a) Rice grain Image **(b)** Thresholding **(c)** Eroting **(d)** Dilate



(e) Twilight colormap

Figure 7: Rice grain counting by connected component analysis

Question 3

(a) Zooming by nearest neighbor interpolation

This algorithms starts by generating an empty array with zeros with the target image size. Then for each pixel in the empty image, its neighboring pixel in the original image is calculated by dividing the pixel value and rounding it to nearest integer. That neighbor value is assigned as the value for the pixel.

```
def zoom_nearest(img, factor):
   channels = len(img.shape)
   if channels == 2:
       row, col = img.shape
       new_row, new_col = int(row * factor), int(col * factor)
       zoomed_img = np.zeros((new_row, new_col), dtype=np.uint8)
   elif channels == 3:
       row, col, channel = img.shape
       new_row, new_col = int(row * factor), int(col * factor)
       zoomed_img = np.zeros((new_row, new_col, channel), dtype = np.uint8)
   for i in range(new_row):
       for j in range(new_col):
           x, y = int(np.round(i/factor)), int(np.round(j/factor))
           if x>= row :
               x = row-1
           if y >= col:
               y = col - 1
           assert(x < row and y < col)
           zoomed_img[i][j] = img[x][y]
   return zoomed_img
```

The algorithm is tested by calculating sum of squared difference (SSD). The results are interpreted here in the following table. For comparison purposes opencv's implementation of nearest neighbor interpolation is also included

(b) Zooming by Bilinear Interpolation

Bilinear interpolation algorithm also uses the same initial steps as the Nearest neighbor interpolation. But instead of assigning values of neighboring pixels, value of the pixel is calculated by using four diagonal points and bilinear interpolation. he algorithm implemented here has a bug, it does not process the last scalingFactor-1 rows and coloumns hence having a poor ssd per pixel.

```
   def zoom_bilinear(img, factor):
      channels = len(img.shape)
      if channels == 2:
         row, col = img.shape
         new_row, new_col = int(row * factor), int(col * factor)
         zoomed_img = np.zeros((new_row, new_col), dtype=np.uint8)
      elif channels == 3:
         row, col, channel = img.shape
```

```
    new_row, new_col = int(row * factor), int(col * factor)
    zoomed_img = np.zeros((new_row, new_col, channel), dtype=np.uint8)

for i in range(new_row):
    for j in range(new_col):
        x_f, y_f = (i)/factor  , (j)/factor
        x, y = int(x_f), int(y_f)
        x_plus1, y_plus1 = min(x+1, row-1),  min(y+1, col-1)
        x_diff, y_diff = x_f - x, y_f- y

        f1 = np.array(img[x][y])
        f2 =  np.array(img[x_plus1][y])
        f3 =  np.array(img[x][y_plus1])
        f4 =  np.array(img[x_plus1][y_plus1])

        f_y1 = (1. - x_diff) * f1 + x_diff * f2
        f_y2 = (1. - x_diff) * f3 + x_diff * f4
        f_xy = (1. - y_diff) * f_y1 + y_diff * f_y2
        zoomed_img[i][j] = (f_xy+0.5).astype('uint8')
return zoomed_img
```

Images	Nearest Neighbor	opencv nearest	Bilinear	opencv bilinear
image1	40.11	31.28	39.39	31.05
image2	16.79	11.90	16.31	10.68
image4	81.75	78.73	81.61	79.30
image5	54.60	50.57	53.79	50.78
image6	34.94	30.55	35.74	31.78
image7	30.57	27.96	30.42	28.564
image9	28.08	21.14	26.85	19.14

Table 1: Comparison of Nearest neighbor interpolation zooming. Values presented are SSD per pixel values. i.e. SSD/image.size

From the results of the table 1 it can be seen that bilinear interpolation performs better than nerest neighbour interpolation. But opencv's builtin functions are optimized by a far amount hence the low ssd values are present. Bilinear interpolation will easily best nearest neighbor when the scaling factor gets larger. For larger scaling factors nearest neighbor will produce blurred images.

Appendix

The entire work of this assignment is uploaded as a notebook in this link `https://github.com/thieshanthan/EN2550_assignment_a01`.