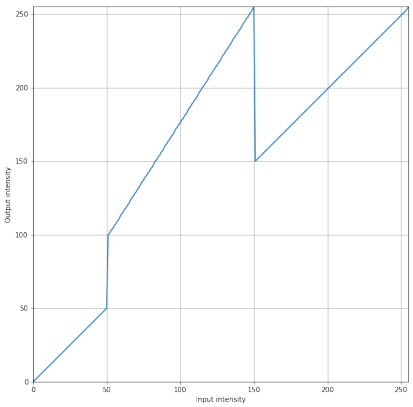
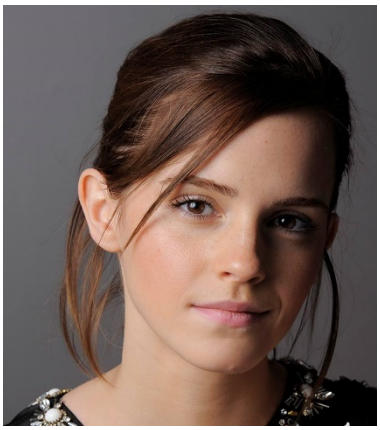
200687M Vishwamith P.G.H Assignment 1

1)



Intensity Transformation



Original Image

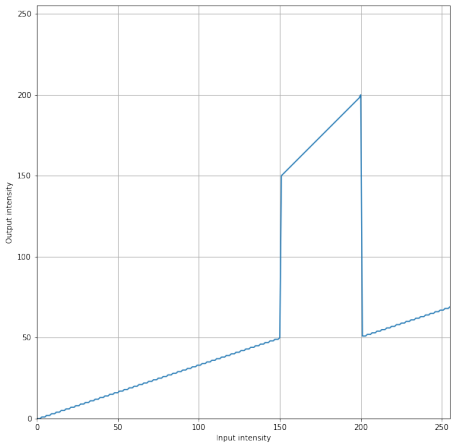
Transformed Image

```
t1 = np.linspace(0, 50, 51).astype('uint8')
t2 = np.linspace(100, 255, 100).astype('uint8')
t3 = np.linspace(150, 255, 255 - 150).astype('uint8')
```

```
transform = np.concatenate((t1, t2), axis=0).astype('uint8')
transform = np.concatenate((transform, t3), axis=0).astype('uint8')
```

Pixel with Intensity between 150 – 50 have enhanced. Other Intensity values haven't changed since it's a y=x line.

2)



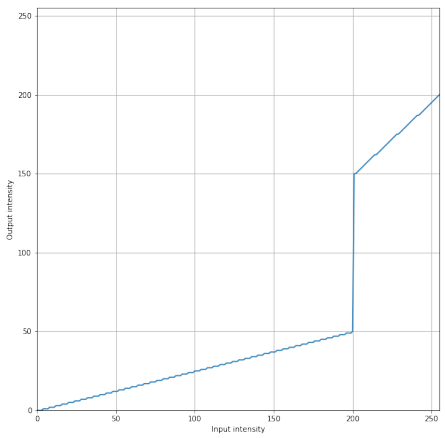
Gary matter enhancing transform

Gary matter enhanced image

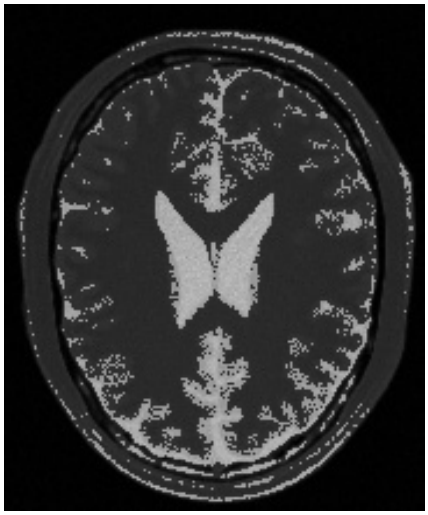
t1 = np.linspace(0, 50, 150-0+1).astype('uint8')

t2 = np.linspace(150, 200, 200-150).astype('uint8')

t3 = np.linspace(51, 69, 255 - 200).astype('uint8')



White matter enhancing transform



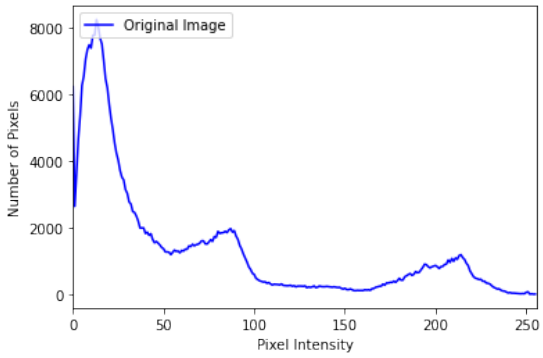
White matter enhanced image

t1 = np.linspace(0, 50, 200-0+1).astype('uint8')

t2 = np.linspace(150, 200, 255-200).astype('uint8')

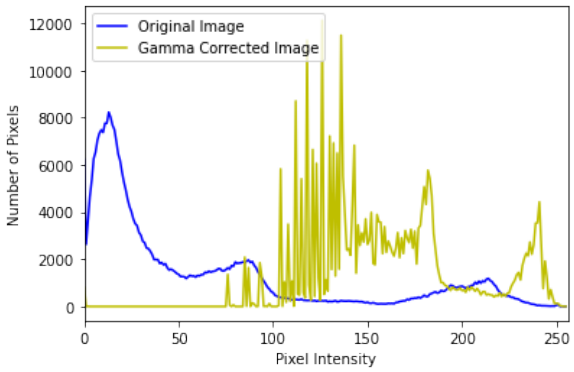
3)

Original image



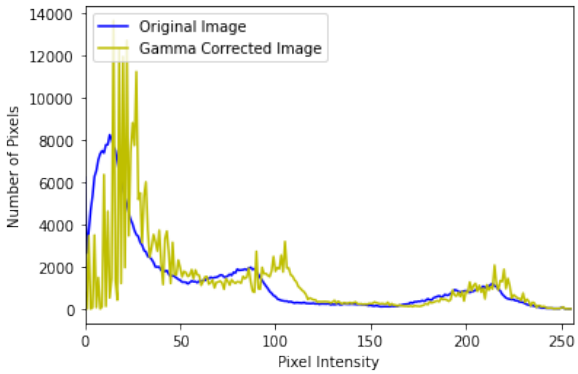
Histogram

Gamma = 0.2



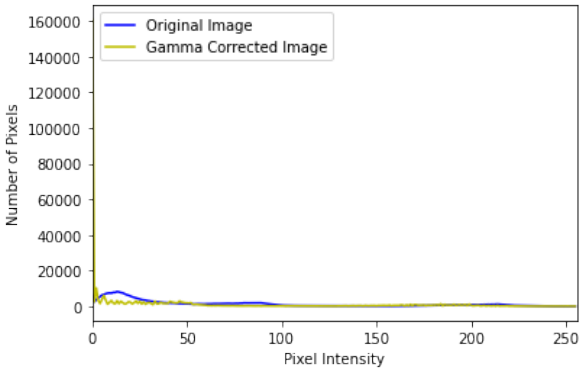
Histogram for Gamma = 0.2

Gamma = 0.8



Histogram for Gamma = 0.8

Gamma = 2



Histogram for Gamma = 2

The 0.8 is the best choice for gamma value. Image gets brighter for gamma >1 and daker for gamma <1. Gamma values between 0 and 1 expand the range of darker pixels in an image, making it appear brighter by shifting colors that were originally in the middle towards the lighter end while keeping black and white unchanged. Conversely, gamma values greater than 1 have the opposite effect, compressing the range of darker pixels and making the image appear darker.

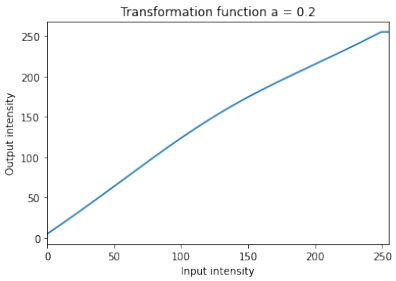
```python
# convert to "L*a*b*" color space
img_lab = cv.cvtColor(img_orig, cv.COLOR_BGR2LAB)

gamma = 1.6
table = np.array([(i/255.0)**(gamma)*255.0 for i in np.arange(0, 256)]).astype('uint8')
L_img = cv.LUT(img_lab[:,:,0], table).reshape(img_lab.shape[0], img_lab.shape[1], 1)
corrected_img=np.concatenate([L_img, img_lab[:,:,1:]], axis=-1)

img_gamma = cv.cvtColor(corrected_img, cv.COLOR_LAB2BGR)
```

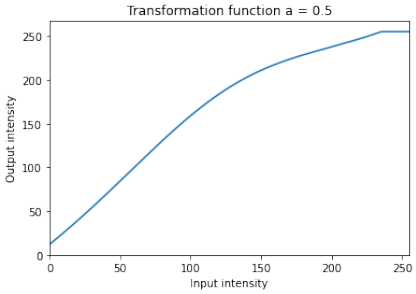
4)

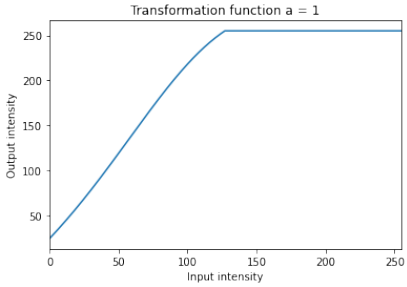
Value image

Hue image

Saturation image





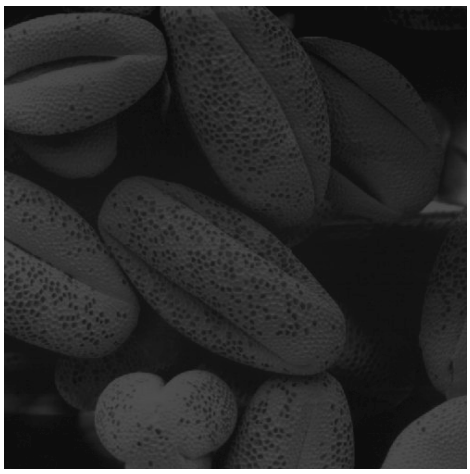


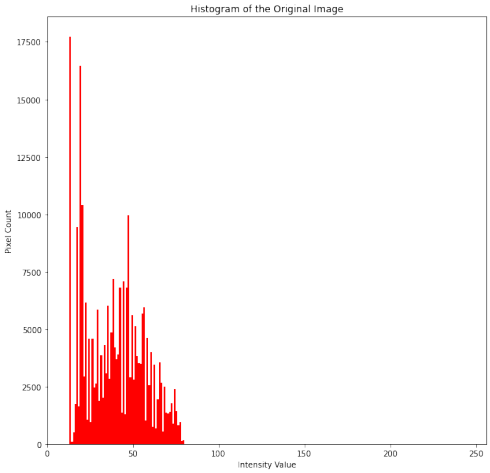
Boosting an image's vibrance involves intensifying the less vibrant colors while keeping the already saturated ones as they are. More precisely, vibrance accentuates subdued colors, typically paying less attention to warmer hues like yellows, oranges, and reds and instead emphasizing cooler colors like blues and greens. The suitable alpha value is 0.5

```python
alpha = 1
def f(x): return min(x + alpha*128*math.exp(-(x-128)**2/(2*70**2)), 255)
image = cv.imread('./Images/spider.png')
hsv_image = cv.cvtColor(image, cv.COLOR_BGR2HSV)
hue, saturation, value = cv.split(hsv_image)
transformed_saturation = np.vectorize(f)(saturation)
transformed_saturation = np.round(transformed_saturation).astype('uint8')
hsv_image[:, :, 1] = transformed_saturation
transformed_image = cv.cvtColor(hsv_image, cv.COLOR_HSV2BGR)
```

Original image

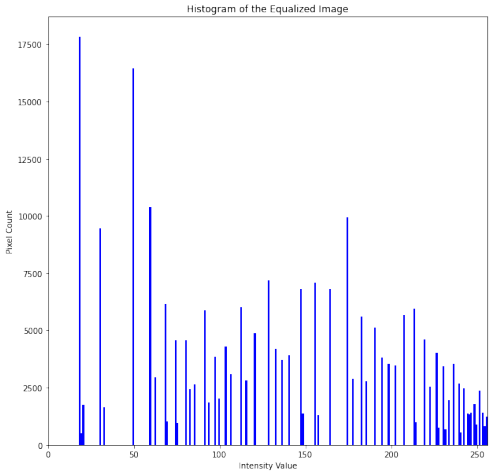
5)





Here are the steps carried out within the function:

1. Calculate the cumulative distribution function (CDF) of the histogram generated.
2. Normalize the cumulative sum.
3. Map equalized pixel intensities to the image pixel intensities using a Look Up table




```
hist = cv.calcHist([image],[0],None,[256],[0,256])
sum_n = np.cumsum(hist)
equalized_r = np.round((255*sum_n)/(image.shape[0]*image.shape[1])).astype('uint8')
equalized_img = cv.LUT(image, equalized_r)
```

6)

Hue image

Value image

Saturation image

Foreground Only image

Foreground Equalized Image

Processed Image

```
thresh1 = cv.threshold(saturation, 11, 255, cv.THRESH_BINARY)[1]
for_grnd = cv.bitwise_and(value, thresh1)
equ = cv.equalizeHist(for_grnd)
bak_grnd = value.copy()-for_grnd.copy()
processed_img = equ.copy()+bak_grnd.copy()
```

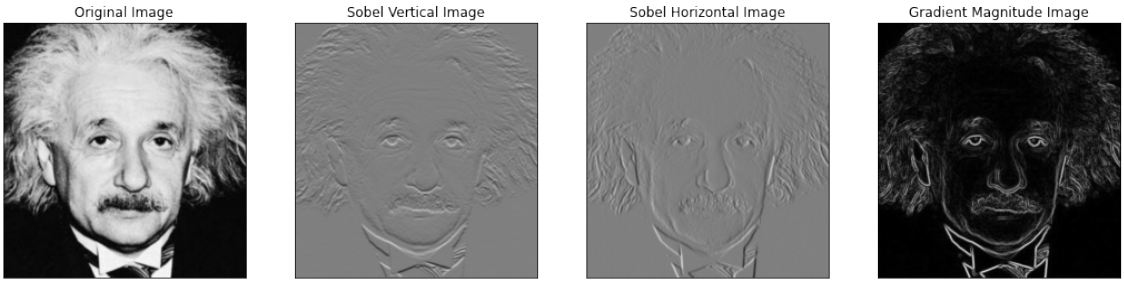
Generate a threshold to remove the background using bit wise and, then get the background by subtracting foreground from grayscale image. Then add the equalized image to the background.

Cumulative sum

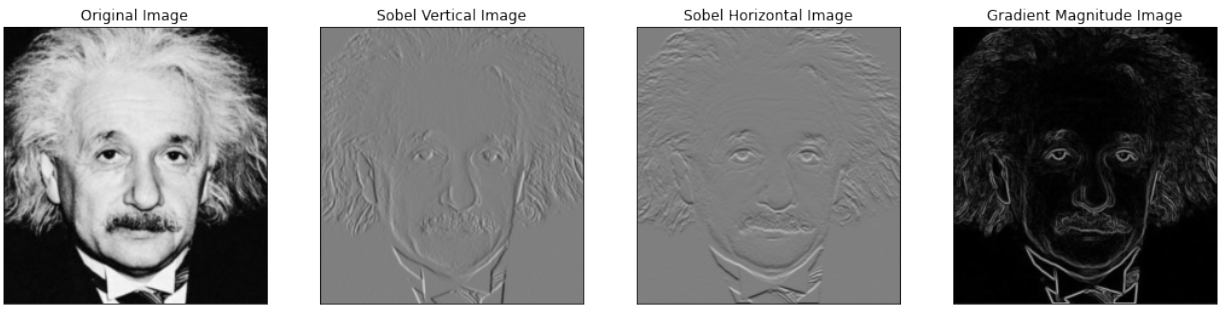
```
array([222621, 222630, 222641, 222659, 222684, 222772, 222806, 222842,
       222899, 222946, 222995, 223063, 223157, 223241, 223332, 223448,
       223603, 223786, 223952, 224173, 224384, 224592, 224844, 225070,
```

7)

```
# Sobel Vertical Kernel
sobel_vertical_kernel = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]], dtype=np.float32)
vertical_gradient = cv.filter2D(input_image, -1, sobel_vertical_kernel)

# Sobel Horizontal Kernel
sobel_horizontal_kernel = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], dtype=np.float32)
horizontal_gradient = cv.filter2D(input_image, -1, sobel_horizontal_kernel)

# Gradient Magnitude Kernel
gradient_magnitude = np.sqrt(vertical_gradient**2 + horizontal_gradient**2)
```



Creating own Sobel filter

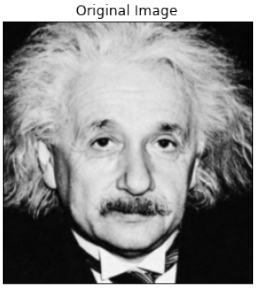


```python
# Sobel Vertical Kernel
sobel_vertical_kernel = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]], dtype=np.float32)
sobel_x = cv.filter2D(input_image, -1, sobel_vertical_kernel)

# Sobel Horizontal Kernel
sobel_horizontal_kernel = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], dtype=np.float32)
sobel_y = cv.filter2D(input_image, -1, sobel_horizontal_kernel)

img_gradient_x = np.zeros(input_image.shape)
img_gradient_y = np.zeros(input_image.shape)
rows, columns = input_image.shape

# Carry out padding
padding_value = 0
padded_image = np.full((rows + 2, columns + 2), padding_value, dtype=np.uint8)

# Copy input_image into the center of the padded image
padded_image[1:rows + 1, 1:columns + 1] = input_image

for i in range(rows):
    for j in range(columns):
        img_gradient_x[i, j] = np.sum(np.multiply(sobel_horizontal_kernel, padded_image[i:i + 3, j:j + 3]))

for i in range(rows):
    for j in range(columns):
        img_gradient_y[i, j] = np.sum(np.multiply(sobel_vertical_kernel, padded_image[i:i + 3, j:j + 3]))

# Calculate the Gradient Magnitude
img_gradient_magnitude = np.sqrt(img_gradient_x**2 + img_gradient_y**2)
```

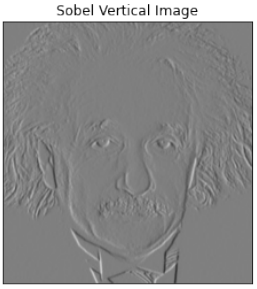
Using the property to make the Sobel filter

```python
# Define the Sobel Vertical Kernel
sobel_ver_kernel1 = np.array([[1], [2], [1]])
sobel_ver_kernel2 = np.array([[1, 0, -1]])

# Define the Sobel Horizontal Kernel
sobel_hor_kernel1 = np.array([[1], [0], [-1]])
sobel_hor_kernel2 = np.array([[1, 2, 1]])

# Initialize intermediate images
img_xp_1 = np.zeros(input_image.shape)
img_xp = np.zeros(input_image.shape)
img_yp_1 = np.zeros(input_image.shape)
img_yp = np.zeros(input_image.shape)

# Perform convolution using scipy.signal
img_xp_1 = sig.convolve2d(input_image, sobel_ver_kernel1, mode="same")
img_xp = sig.convolve2d(img_xp_1, sobel_ver_kernel2, mode="same")
img_yp_1 = sig.convolve2d(input_image, sobel_hor_kernel1, mode="same")
img_yp = sig.convolve2d(img_yp_1, sobel_hor_kernel2, mode="same")

# Calculate the Gradient Magnitude
img_grad_p = np.sqrt(img_xp**2 + img_yp**2)
```

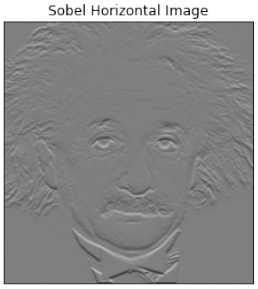
Original Image



Sobel Vertical Image



Sobel Horizontal Image



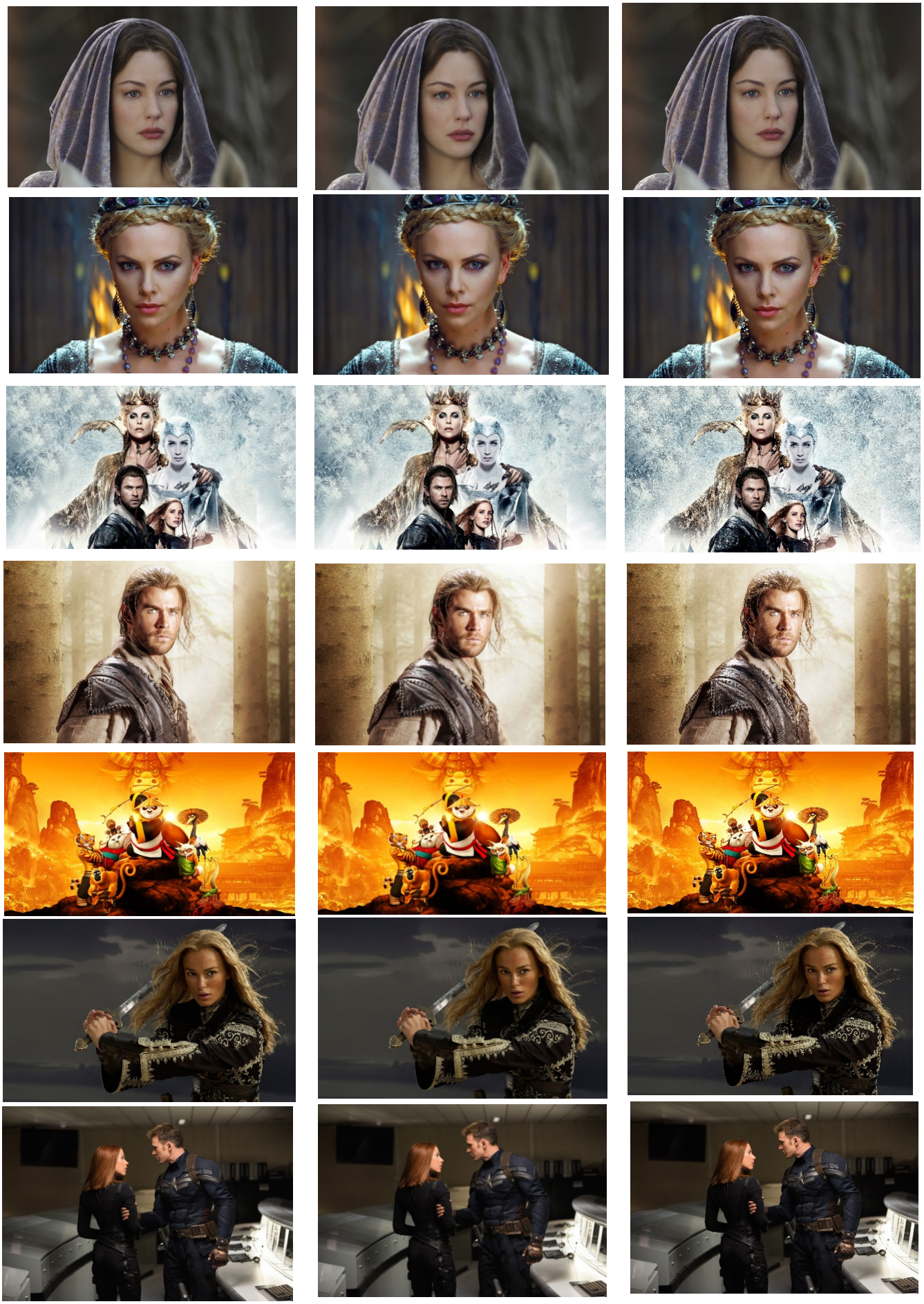
Gradient Magnitude Image

Nearest Neighbor

Bilinear Interpolation

Original large

8)



```
resized_image1 = cv.resize(small_image.copy(), (width, height), interpolation=cv.INTER_NEAREST)
resized_image2 = cv.resize(small_image.copy(), (width, height), interpolation=cv.INTER_LINEAR )
ssd_val_ner = np.sum(((original_image[:,:]-resized_image1[:,:])**2)/(3*255**2))/(original_image.shape[0]*original_image.shape[1])
ssd_val_biln = np.sum(((original_image[:,:]-resized_image2[:,:])**2)/(3*255**2))/(original_image.shape[0]*original_image.shape[1])
```

When using Bilinear Interpolation. The Images get a lot smoother

Table 1: SSD values

Image	SSD in Nearest Neighbor	SSD in Bilinear Interpolation
1	0.0004811121335890121	0.0004775562417232688
2	0.0001830374980420635	0.0001642905306185653 8
3	size mismatch	
4	0.0012108853093757068	0.0012195573249169796
5	0.0007778123197808548	0.0007810691067004153
6	0.0004698807527731496	0.0004888167863974442
7	0.00043005217469442824	0.00043928369146882653
8	size missmatch	
9	0.0003252303067196373	0.0002943567591524629 6
10	size missmatch	
11	size missmatch	

9)

When using grabCut, there can be pixels near the image's edges that potentially belong to both the foreground and background. As a result, when the two images are combined, these edge pixels are added together giving a value higher than 255. These values are automatically mapped back to zero by the OpenCV library, causing them to appear darker.

https://github.com/HirunaVishwamith/Image-processing-assignmnet-1

```
mask = np.zeros(img.shape[:2], np.uint8)
bgdModel = np.zeros((1, 65), np.float64)
fgdModel = np.zeros((1, 65), np.float64)
rect = (50, 100, 450, 450)
cv.grabCut(img, mask, rect, bgdModel, fgdModel, 5, cv.GC_INIT_WITH_RECT)
maskforg = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')
maskbkg = np.where((mask == 2) | (mask == 0), 1, 0).astype('uint8')
themask = np.where((mask == 2) | (mask == 0), 0, 255).astype('uint8')
imgf = img * maskforg[:, :, np.newaxis]
imgb = img * maskbkg[:, :, np.newaxis]
imgb = cv.GaussianBlur(imgb, (5, 5), 10)
new_img = imgf + imgb
```