In [1]:
```python
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from scipy.signal import convolve2d
import math
```

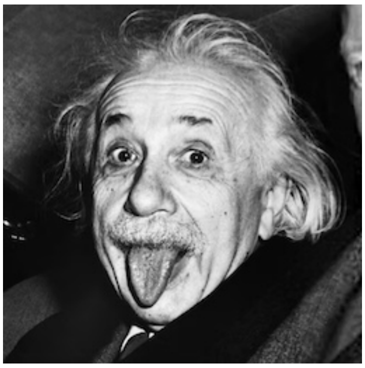
Homework 1

(A) Choosing an example image

In [2]:
```python
def read_image(file):
    image = Image.open(file)

    # Convert to grayscale
    grey_avg_array = (np.sum(image, axis = -1, keepdims = False) / 3)
    grey_avg_array = grey_avg_array.astype(np.uint8)
    image = Image.fromarray(grey_avg_array, "L")

    return image
```

In [3]:
```python
image = read_image("resources/einstein.jpeg")
image
```

Out[3]:



(B) Simple center-surround receptive field

Filter the photo by the center-surround receptive fields of the retinal ganglion cells. You can start with the practice of the toy models of the center-surround receptive fields that we used in the tutorial. These toy models have receptive field's shape described by $K(x, y)$ as a function of horizontal and vertical displacement x and y from the center of the receptive field.

$$K(x, y) = \begin{cases} 1 & \text{when } |x| < L/2, |y| < L/2 \\ -v & \text{when } |x| \geq L/2, |y| \geq L/2 \end{cases}$$

Give a couple of examples using different parameters L and v. Plot out the responses of the ganglion cells as an image to show the outcomes.

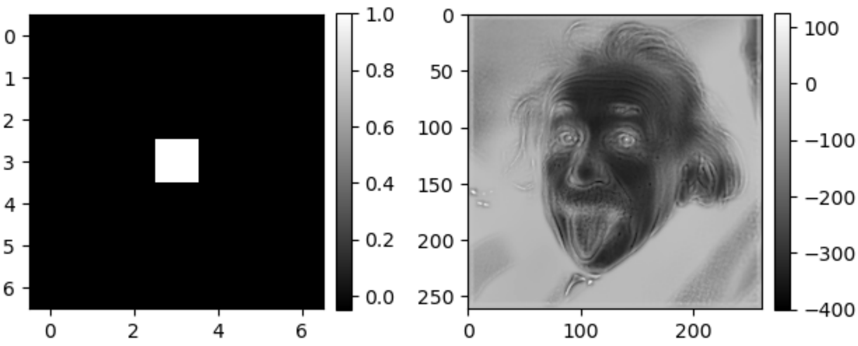
In [4]:
```python
def square_filter(v, l):
    filter = np.ones(shape = (l, l)) * v

    # Estimate the center of the receptive field
    center = int(l / 2)

    # Estimate the width of the excitatory receptive field area
    width = int(l / 4)
    if width == 0: width = 1

    for i in range(filter.shape[0]):
        for j in range(filter.shape[1]):
            if np.abs(i - center) < width and np.abs(j - center) < width:
                filter[i, j] = 1

    return filter
```

In [5]:
```python
def apply_filter(image, filter):
    return convolve2d(image, filter)
```

In [6]:
```python
def plot_filter_and_image(filter, image):
    plt.subplot(1, 2, 1)
    plt.imshow(filter, cmap = "gray")
    plt.colorbar(fraction = 0.046, pad = 0.04)

    plt.subplot(1, 2, 2)
```

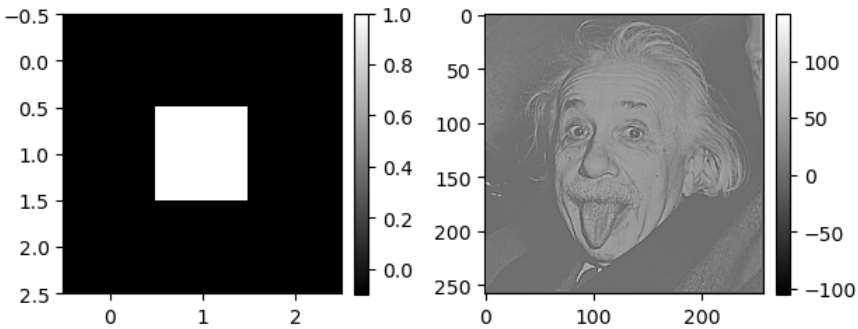
```
    plt.imshow(image, cmap = "gray")
    plt.colorbar(fraction = 0.046, pad = 0.04)

    plt.tight_layout()
    plt.show()
```

In [7]:
```
filter = square_filter(v = -0.05, l = 7)
plot_filter_and_image(filter, apply_filter(image, filter))
```



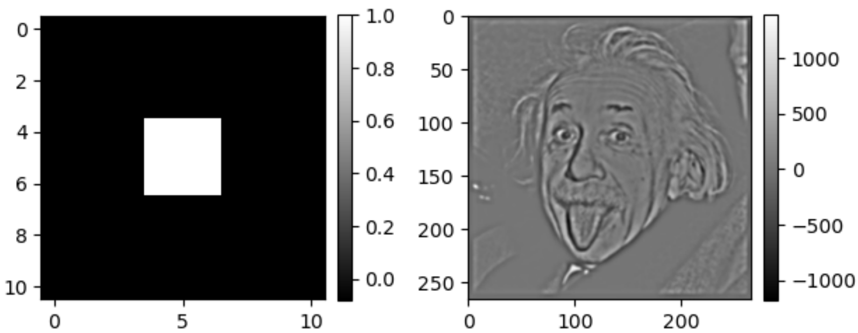
The image is filtered using $L = 7$ and $v = -0.05$. The colors are inverted because the average filter value is negative. Also, large receptive field makes the image more blurry.

In [8]:
```
filter = square_filter(v = -0.1, l = 3)
plot_filter_and_image(filter, apply_filter(image, filter))
```



The image is filtered using $L = 3$ and $v = -0.1$. This time, the average filter value is positive and the colors are not inverted. Using a smaller receptive field maintains the acuity.

In [9]:
```
filter = square_filter(v = -9 / (11 * 11 - 9), l = 11)
plot_filter_and_image(filter, apply_filter(image, filter))
```



The image is filtered using $L = 11$ and $v \approx -0.08$. This time, the v parameter calculated so that the average filter value is close to zero which results in non-inverted colors. This filter is able to detect edges within the image.

(C) Difference-of-Gaussians filter

Repeat (B) using a difference-of-gaussian $K(x, y)$ as

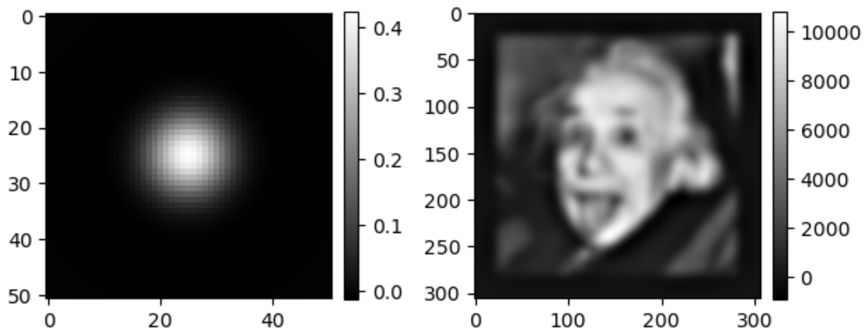
$$K(x, y) = \frac{w_c}{\sigma_c^2} exp(-\frac{x^2 + y^2}{2\sigma_c^2}) - \frac{w_s}{\sigma_s^2} exp(-\frac{x^2 + y^2}{2\sigma_s^2})$$

Play with parameters w_c, w_s, σ_c and σ_s and understand how $K(x, y)$ changes with each these parameters, and thus the meaning of these parameters. Filter the original image using this $K(x, y)$ and see the outcome as the retinal ganglion cells population responses. For a retinal ganglion cell, you may try $w_c = 1.1 w_s$ and $\sigma_s = 5\sigma_c$.

In [10]:
```
def gaussian(w_center, w_surround, sigma_center, sigma_surround, x, y):
    return (w_center / math.pow(sigma_center, 2)) * np.exp(-(math.pow(x, 2) + math.pow(y, 2)) / (2 * math.pow(s
        (w_surround / math.pow(sigma_surround, 2)) * np.exp(-(math.pow(x, 2) + math.pow(y, 2)) / (2 * math.pow(
```

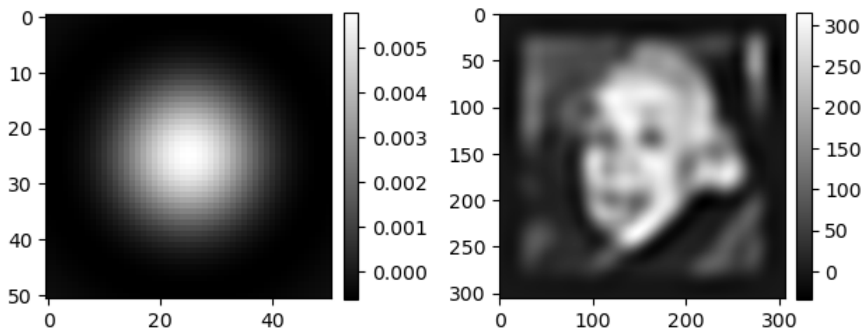
In [11]:
```
def gaussian_filter(w_center, w_surround, sigma_center, sigma_surround, l = 101):
    filter = np.zeros(shape = (l, l))

    center = int(l / 2)

    for i in range(filter.shape[0]):
        for j in range(filter.shape[1]):
            filter[ i, j ] = gaussian(w_center, w_surround, sigma_center, sigma_surround, i - center, j - cente

    return filter
```

In [12]:
```
filter = gaussian_filter(
    w_center = 11, # center strength
    sigma_center = 5, # center radius
    w_surround = 10, # surround strength
    sigma_surround = 25, # surround radius
    l = 51
)

plot_filter_and_image(filter, apply_filter(image, filter))
```



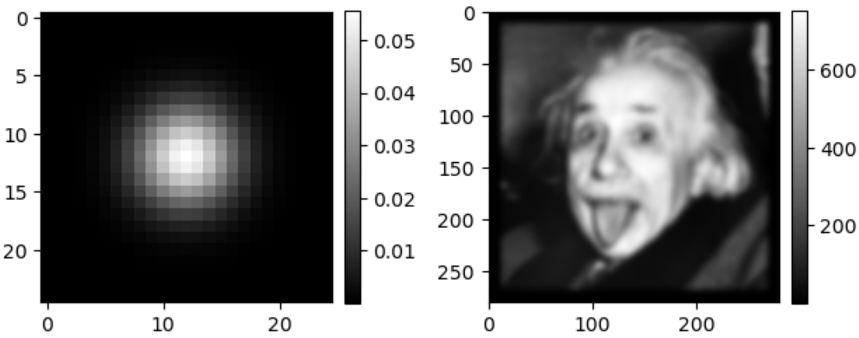
The image is filtered using $w_c = 11$, $w_s = 10$, $\sigma_c = 5$ and $\sigma_s = 25$. The w_c and w_s parameters control the **strength** of the center and the surround Gaussian, accordingly. The σ_c and σ_s parameters control the **size** of the center and the surround Gaussian, accordingly.

In [13]:
```
filter = gaussian_filter(
    w_center = 1, # center strength
    sigma_center = 10, # center radius
    w_surround = 0.95, # surround strength
    sigma_surround = 15, # surround radius
    l = 51
)

plot_filter_and_image(filter, apply_filter(image, filter))
```



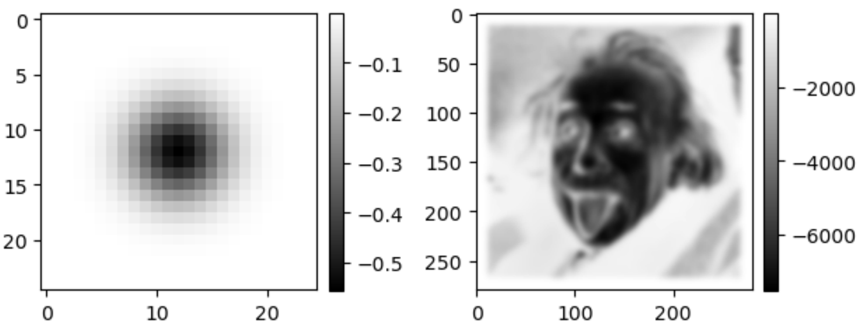
The image is filtered using $w_c = 1$, $w_s = 0.95$, $\sigma_c = 10$ and $\sigma_s = 15$.

In [14]:
```
filter = gaussian_filter(
    w_center = 0.5, # center strength
    sigma_center = 3, # center radius
    w_surround = 0, # surround strength
    sigma_surround = 1, # surround radius
    l = 25
)

plot_filter_and_image(filter, apply_filter(image, filter))
```



The image is filtered using $w_c = 0.5$, $w_s = 0$, $\sigma_c = 3$ and $\sigma_s = 1$ (only the center Gaussian is used).

In [15]:
```
filter = gaussian_filter(
    w_center = 0, # center strength
    sigma_center = 1, # center radius
    w_surround = 5, # surround strength
    sigma_surround = 3, # surround radius
    l = 25
)

plot_filter_and_image(filter, apply_filter(image, filter))
```



The image is filtered using $w_c = 0$, $w_s = 5$, $\sigma_c = 1$ and $\sigma_s = 3$ (only the surround Gaussian is used).

(D) Vertical orientation selective neuron

Repeat (C), but use a receptive field filter shape $K(x, y)$ that models a V1's simple cell's receptive field. Use an orientation selective neuron (tuning to vertical orientation), with

$$K(x, y) = exp(-\frac{x^2}{2\sigma_x^2} - \frac{y^2}{2\sigma_y^2})cos(\bar{k}x + \phi)$$

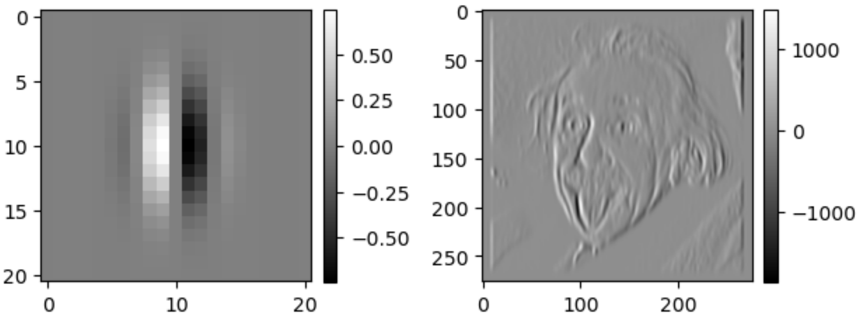
Vary parameters σ_x, σ_y, $\bar{k}$, and ϕ and plot out $K(x, y)$ using different set of parameters to see how these parameters control the shape of $K(x, y)$. For a V1 cell model, try for example $\sigma_y = 1.5\sigma_x$, and $\bar{k} = 2\pi/(3\sigma_x)$, and filter the original image using this $K(x, y)$ and see the outcome as the population responses of V1 neurons preferring this orientation.

In [16]:
```
def vertical_orientation(sigma_x, sigma_y, k, phi, x, y):
    return np.exp(-(math.pow(x, 2) / (2 * math.pow(sigma_x, 2))) - (math.pow(y, 2)) / (2 * math.pow(sigma_y, 2)
```

In [17]:
```
def vertical_filter(sigma_x, sigma_y, k, phi, l = 101):
    filter = np.zeros(shape = (l, l))

    center = int(l / 2)

    for i in range(filter.shape[0]):
        for j in range(filter.shape[1]):
            filter[i, j] = vertical_orientation(sigma_x, sigma_y, k, phi, j - center, i - center)
```

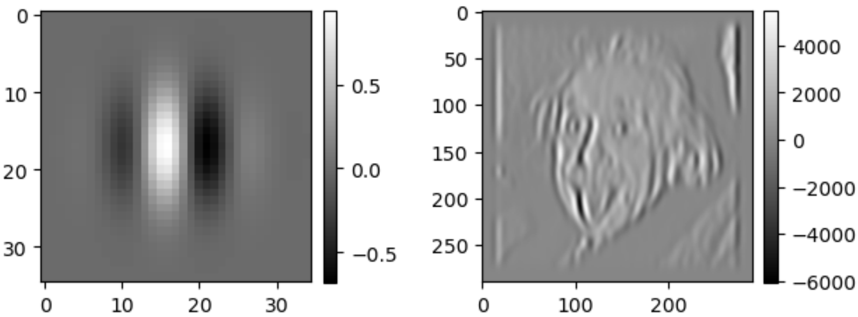
```python
    return filter
```

In [18]:
```python
filter = vertical_filter(
    sigma_x = 2,
    sigma_y = 3,
    k = 1,
    phi = math.radians(90),
    l = 21
)

plot_filter_and_image(filter, apply_filter(image, filter))
```



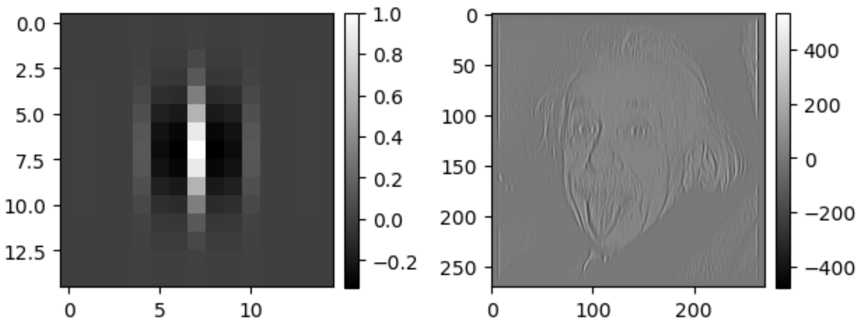
The image is filtered using $\sigma_x = 2$, $\sigma_y = 3$, $\bar{k} = 1$, and $\phi = 90^\circ$.

In [19]:
```python
filter = vertical_filter(
    sigma_x = 5,
    sigma_y = 5,
    k = 0.5,
    phi = math.radians(45),
    l = 35
)

plot_filter_and_image(filter, apply_filter(image, filter))
```



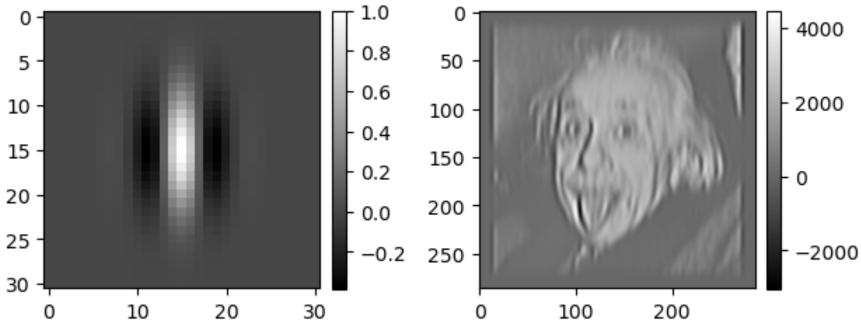
The image is filtered using $\sigma_x = 5$, $\sigma_y = 5$, $\bar{k} = 0.5$, and $\phi = 45^\circ$.

In [20]:
```python
filter = vertical_filter(
    sigma_x = 1.5,
    sigma_y = 2,
    k = 2,
    phi = math.radians(0),
    l = 15
)

plot_filter_and_image(filter, apply_filter(image, filter))
```



The image is filtered using $\sigma_x = 1.5$, $\sigma_y = 2$, $\bar{k} = 2$, and $\phi = 0^\circ$.

In [21]:
```
filter = vertical_filter(
    sigma_x = 3,
    sigma_y = 3 * 1.5,
    k = (2 * math.pi) / (3 * 3),
    phi = math.radians(0),
    l = 31
)

plot_filter_and_image(filter, apply_filter(image, filter))
```



The image is filtered using $\sigma_x = 3$, $\sigma_y = 1.5\sigma_x$, $\bar{k} = 2\pi/(3\sigma_x)$, and $\phi = 0^{\circ}$.

(E) Horizontal orientation selective neuron

Repeat (D), but using a $K(x, y)$ that is tune to horizontal orientation, i.e.,

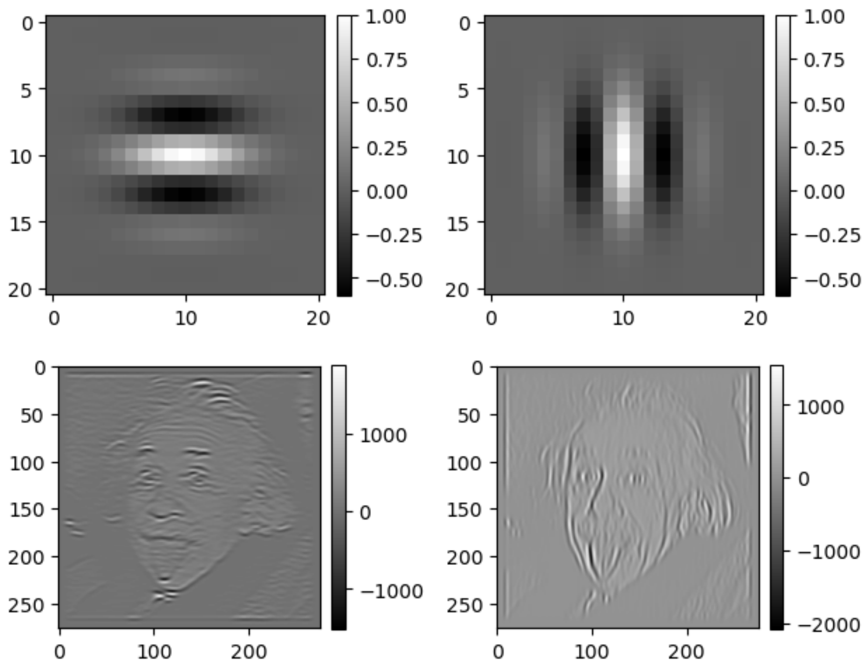
$$K(x, y) = exp(-\frac{x^2}{2\sigma_x^2} - \frac{y^2}{2\sigma_y^2})cos(\bar{k}y + \phi)$$

Plot out $K(x, y)$ from equations 3 and 4 side by side to compare and see they how differ from each other. Plot out the neural population responses from these two filters side by side and see how they differ from each other, and comment on these differences.

In [22]:
```
def horizontal_orientation(sigma_x, sigma_y, k, phi, x, y):
    return np.exp(-(math.pow(x, 2) / (2 * math.pow(sigma_x, 2))) - (math.pow(y, 2)) / (2 * math.pow(sigma_y, 2)
```

In [23]:
```
def horizontal_filter(sigma_x, sigma_y, k, phi, l = 101):
    filter = np.zeros(shape = (l, l))

    center = int(l / 2)

    for i in range(filter.shape[0]):
        for j in range(filter.shape[1]):
            filter[i, j] = horizontal_orientation(sigma_x, sigma_y, k, phi, j - center, i - center)

    return filter
```

In [24]:
```
filter_horizontal = horizontal_filter(
    sigma_x = 3,
    sigma_y = 3,
    k = 1,
    phi = math.radians(0),
    l = 21
)

filter_vertical = vertical_filter(
    sigma_x = 3,
    sigma_y = 3,
    k = 1,
    phi = math.radians(0),
    l = 21
)

plot_filter_and_image(filter_horizontal, filter_vertical)
plot_filter_and_image(apply_filter(image, filter_horizontal), apply_filter(image, filter_vertical))
```

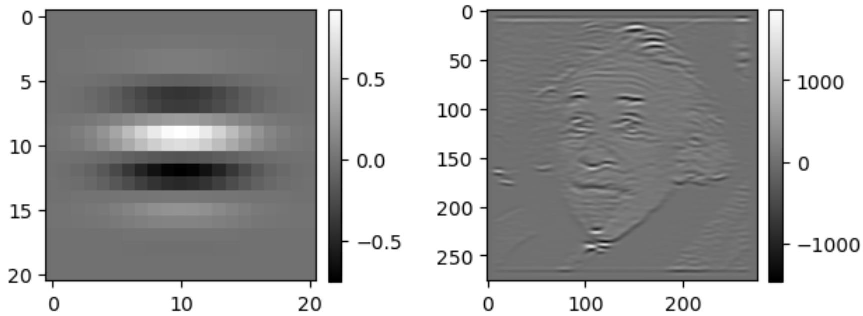


The left image is filtered using a horizontal filter with parameters $\sigma_x = 3$, $\sigma_y = 3$, $\bar{k} = 1$, and $\phi = 0^\circ$.

The right image is filtered using a vertical filter with parameters $\sigma_x = 3$, $\sigma_y = 3$, $\bar{k} = 1$, and $\phi = 0^\circ$.

The two filters are rotated 90 degrees in respect to each other. The vertically tuned filter detects vertical lines, and the horizontally tuned filter detects horizontal lines. Both filters blur other features in the picture.

In [25]:
```
filter = horizontal_filter(
    sigma_x = 3.5,
    sigma_y = 3,
    k = 1,
    phi = math.radians(45),
    l = 21
)

plot_filter_and_image(filter, apply_filter(image, filter))
```



The image is filtered using a filter with parameters $\sigma_x = 3.5$, $\sigma_y = 3$, $\bar{k} = 1$, and $\phi = 45^\circ$.

(F) Contrast sensitivity function for a retinal ganglion cell

Use the retinal ganglion cell's $K(x, y)$ from part (C) and try to get its contrast sensitivity function $g(k)$ by calculating

$$g_c(k) = \sum_{x,y} K(x, y) cos(kx)$$

and

$$g_s(k) = \sum_{x,y} K(x, y) sin(kx)$$

Finally, calculate

$$g(k) = \sqrt{[g_s(k)]^2 + [g_c(k)]^2}$$

for all kinds of values of k, so that you get $g(k)$ as a function of k, and plot out $g(k)$ versus k. Do you see that $g(k)$ peaks at a particular k?

In [26]:
```
def g_c(kernel, k):
    res = 0

    for i in range(kernel.shape[0]):
        for j in range(kernel.shape[1]):
            pos = j
            res += kernel[i, j] * np.cos(k * pos)

    return res
```

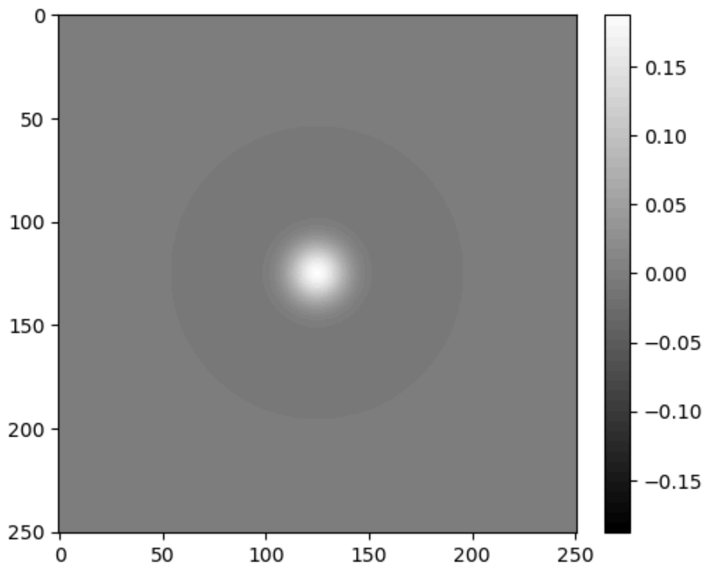
In [27]:
```
def g_s(kernel, k):
    res = 0

    for i in range(kernel.shape[0]):
        for j in range(kernel.shape[1]):
            pos = j
            res += kernel[i, j] * np.cos(k * pos)

    return res
```

In [28]:
```
def g(kernel, k):
    g_k = []
    for _k in k:
        gs = math.pow(g_s(kernel, _k), 2)
        gc = math.pow(g_c(kernel, _k), 2)
        g_k.append(math.sqrt(gs + gc))

    return g_k
```

In [29]:
```
def k_step(pixels):
    return [ n * 2 * math.pi / pixels for n in range(0, int(pixels / 2)) ]
```

In [30]:
```
kernel = gaussian_filter(
    w_center = 20, # center strength
    sigma_center = 10, # center radius
    w_surround = 30, # surround strength
    sigma_surround = 10 * 5, # surround radius
    l = 251
)
```

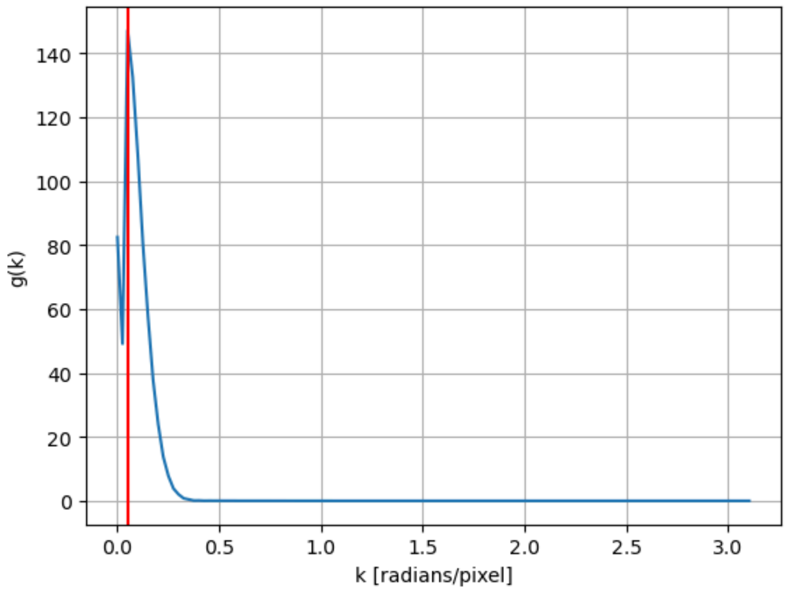
In [31]:
```
plt.imshow(kernel, vmin = -np.max(np.abs(kernel)), vmax = np.max(np.abs(kernel)), cmap = "gray")
plt.colorbar(fraction = 0.046, pad = 0.04)
plt.show()
```



Above is a difference-of-Gaussians filter with parameters $w_c = 20$, $w_s = 30$, $\sigma_c = 10$ and $\sigma_s = 50$.

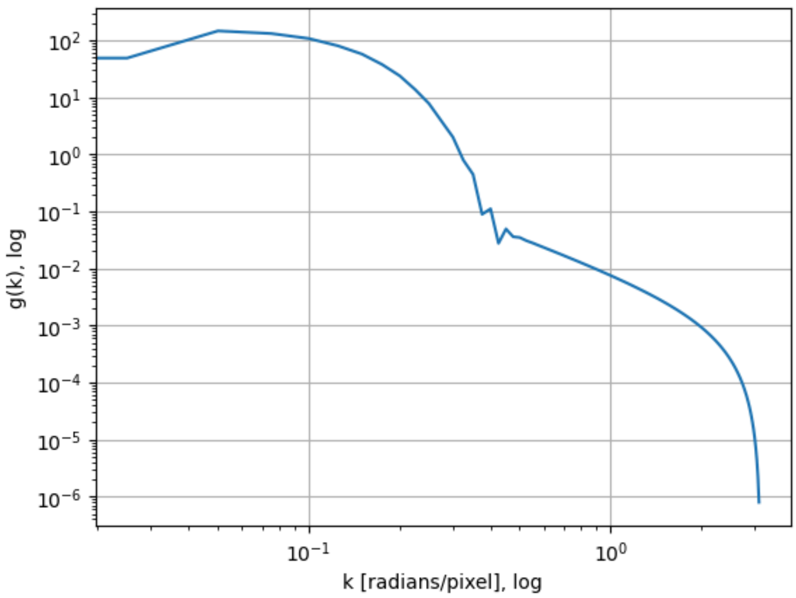
In [32]:
```
k = k_step(251)
g_k = g(kernel, k)
```

In [33]:
```
plt.plot(k, g_k)
plt.axvline(0.05, color = "red")
plt.grid()
plt.xlabel("k [radians/pixel]")
plt.ylabel("g(k)")
plt.show()
```



Contrast sensitivity function of a retinal ganglion cell (difference-of-Gaussians filter) peaks at $k \approx 0.05$.

In [34]:
```
plt.plot(k, g_k)
plt.grid()
plt.xlabel("k [radians/pixel], log")
plt.ylabel("g(k), log")
plt.xscale("log")
plt.yscale("log")
plt.show()
```

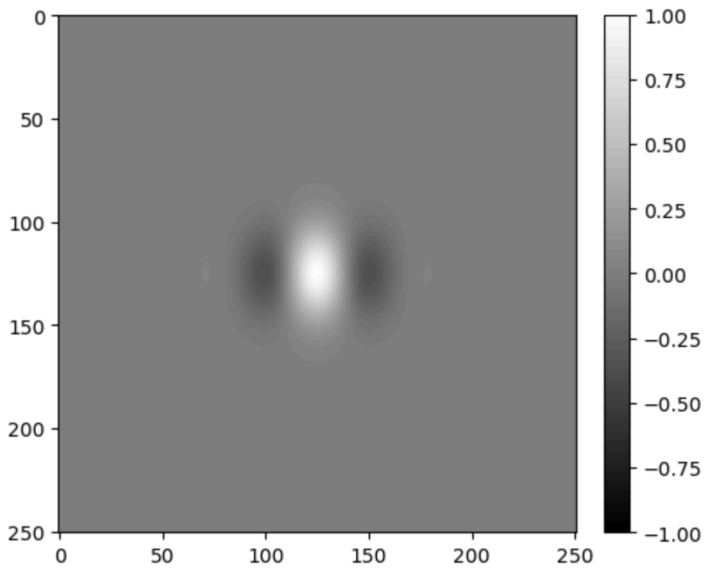


(G) Contrast sensitivity function for V1 cell

Repeat part (F) with V1's receptive field, i.e., use $K(x, y)$ in part (D), and get $g(k)$. Which k value is this $g(k)$ peaking at? Is it around $k = \bar{k}$? Compare this $g(k)$ with the one you have in part (F), and see how they differ from each other.

In [35]:
```
kernel = vertical_filter(
    sigma_x = 20,
    sigma_y = 15,
    k = 0.1,
    phi = math.radians(0),
    l = 251
)
```

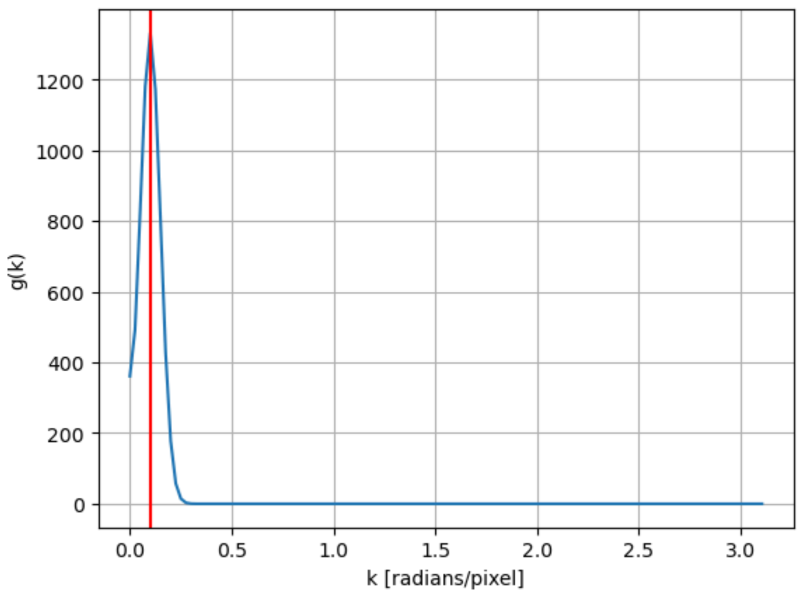
In [36]:
```
plt.imshow(kernel, cmap = "gray", vmin = -np.max(np.abs(kernel)), vmax = np.max(np.abs(kernel)))
plt.colorbar(fraction = 0.046, pad = 0.04)
plt.show()
```



Above is a vertical filter with parameters $\sigma_x = 20$, $\sigma_y = 15$, $\bar{k} = 0.1$, and $\phi = 0^{\circ}$.

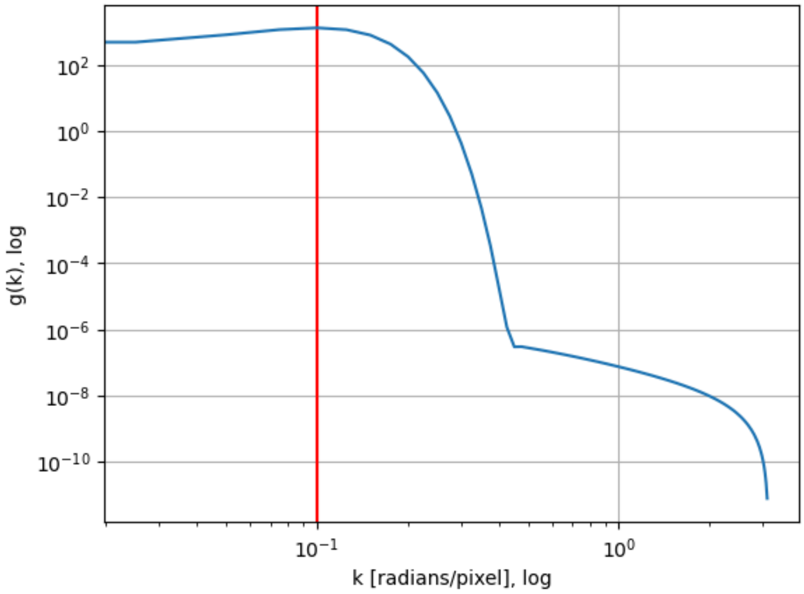
In [37]:
```
k_vertical = k_step(251)
g_k_vertical = g(kernel, k_vertical)
```

In [38]:
```
plt.plot(k_vertical, g_k_vertical)
plt.grid()
plt.axvline(0.1, color = "red")
plt.xlabel("k [radians/pixel]")
plt.ylabel("g(k)")
plt.show()
```

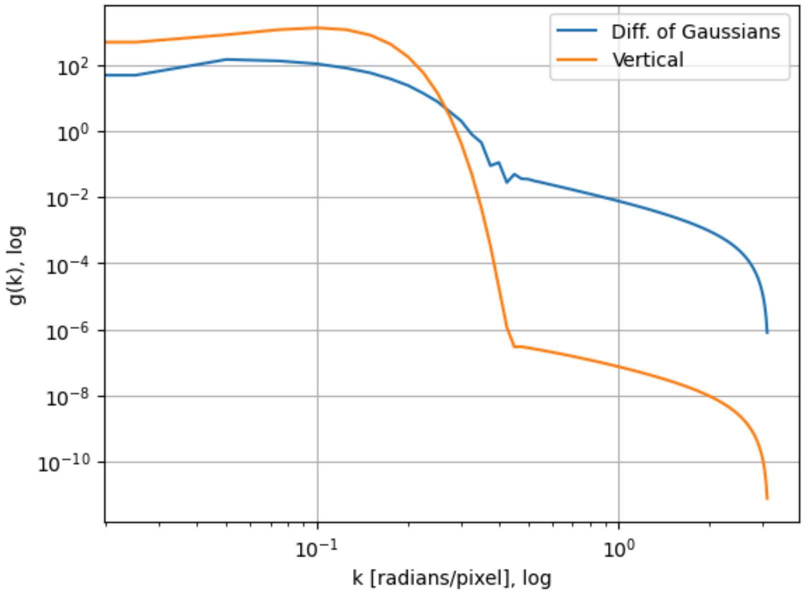


Contrast sensitivity function of a vertical filter peaks at $k \approx 0.1 = \bar{k}$.

In [39]:
```
plt.axvline(0.1, color = "red")
plt.plot(k_vertical, g_k_vertical)
plt.grid()
plt.xlabel("k [radians/pixel], log")
plt.ylabel("g(k), log")
plt.xscale("log")
plt.yscale("log")
plt.show()
```



In [40]:
```
plt.plot(k, g_k, label = "Diff. of Gaussians")
plt.plot(k_vertical, g_k_vertical, label = "Vertical")
plt.grid()
plt.xlabel("k [radians/pixel], log")
plt.ylabel("g(k), log")
plt.xscale("log")
plt.yscale("log")
plt.legend()
plt.show()
```



The contrast sensitivity function of the difference-of-Gaussians filter has a less pronounced drop as k increases. This is due to there being a difference of two Gaussian functions that peak at different values of k. The contrast sensitivity function of the vertical filter, on the other hand, has a stronger peak and then drops fast.

(H) Power spectrum of an image

Repeat part (G) by replacing $K(x, y)$ by your own original image. Let us denote your original image as $S(x, y)$ as a function of x and y.

$$S_c(k) = \sum_{x,y} S(x, y)cos(kx)$$

and

$$S_s(k) = \sum_{x,y} S(x, y)sin(kx)$$

From $S_c(k)$ and $S_s(k)$ calculate

$$|S(k)|^2 = [S_c(k)]^2 + [S_s(k)]^2$$

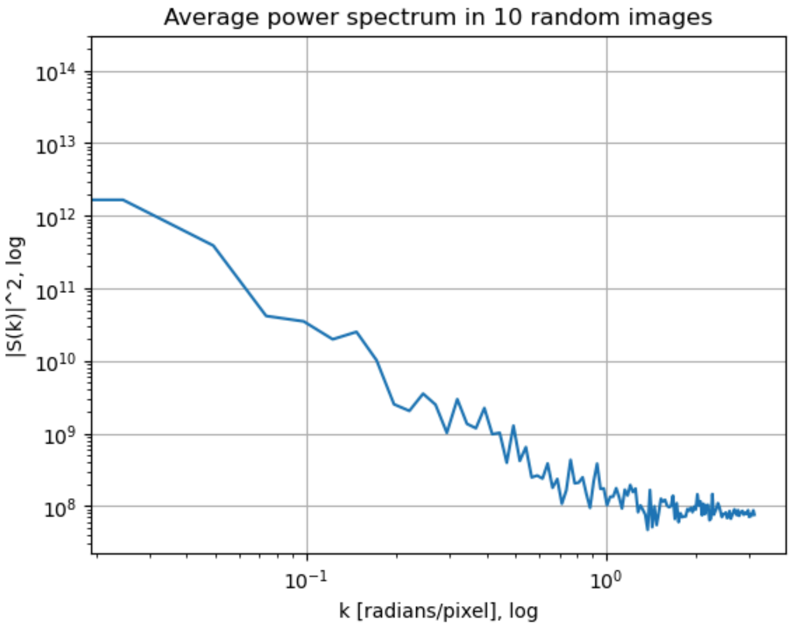
This $|S(k)|^2$ as a function of k is called the power spectrum of an image. Plot out this function, and note a general trend of how $|S(k)|^2$ changes with k.

In [41]:
```
def power_spectrum(image, k):
    g_k = []
    for _k in k:
        gs = math.pow(g_s(image, _k), 2)
        gc = math.pow(g_c(image, _k), 2)
        g_k.append(gs + gc)

    return g_k
```

In [42]:
```
image_files = [ f"resources/image{i}.jpeg" for i in range(1, 10) ]
image_files = np.append(image_files, "resources/einstein.jpeg")

k = k_step(256)
s_k = []

for file in image_files:
    print(f"Processing file {file}")
    image = read_image(file)
    s_k.append(power_spectrum(np.asarray(image, dtype = "uint8"), k))

s_k = np.array(s_k)
```

```
Processing file resources/image1.jpeg
Processing file resources/image2.jpeg
Processing file resources/image3.jpeg
Processing file resources/image4.jpeg
Processing file resources/image5.jpeg
Processing file resources/image6.jpeg
Processing file resources/image7.jpeg
Processing file resources/image8.jpeg
Processing file resources/image9.jpeg
Processing file resources/einstein.jpeg
```

In [43]:
```
plt.plot(k, np.mean(s_k, axis = 0))
plt.grid()
plt.xlabel("k [radians/pixel], log")
plt.ylabel("|S(k)|^2, log")
plt.xscale("log")
plt.yscale("log")
plt.title("Average power spectrum in 10 random images")
plt.show()
```



In general, the power drops as the k increases.

In [43]: