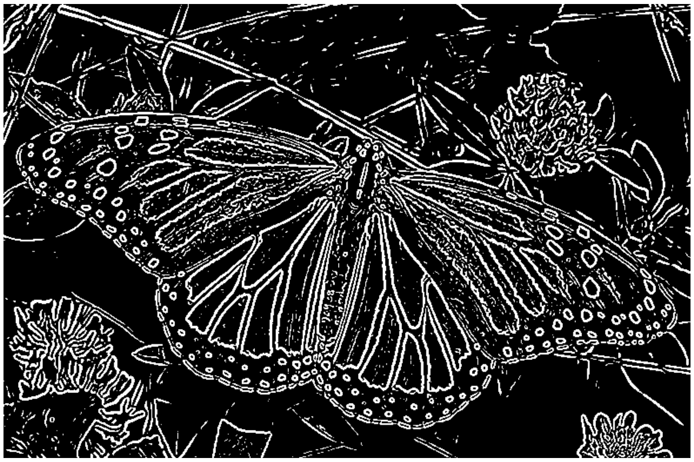
Canny Edge Detection

Canny Edge Detection algorithm is used to detect the edges as the name suggests, and it can be broken down in to multiple steps.



Step 1 The noise of the image has to be reduced, by using smoothening techniques like Gaussian filters or median filters. Here Gaussian filter is used.

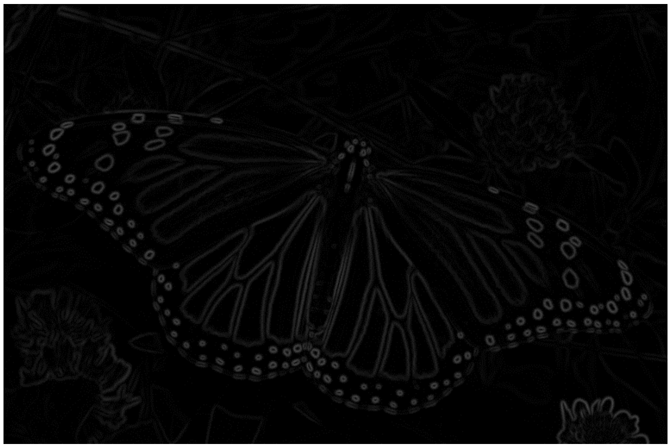
Gaussian matrix: [[1,2,1],[2,4,2],[1,2,1]]

```
def gaussian(I, patch_size):
   patch = [[1,2,1],[2,4,2],[1,2,1]]
      size_i = I.shape
   I1 = np.zeros([size_i[0] - 2, size_i[1]-2])
   for i in range(size_i[0] - 2):
      for j in range(size_i[1]-2):
         output = np.zeros(patch_size)
         for k in range(patch_size[0]):
            for l in range(patch_size[1]):
               output[k,l] = I[k+i, l+j]
         temp = np.sum(patch*output) / 16
         I1[i, j] = temp
```

Step 2 For edge detection, the gradient of the image has to be calculated and this is done using Sobel-x and Sobel-y kernels. Sobel operator finds the first derivative of the Image i.e Ix and Iy.

Sobel-x matrix: [[1,0,-1],[2,0,-2],[1,0,-1]]   Sobel-y matrix: [[1,2,1],[0,0,0],[-1,-2,-1]]

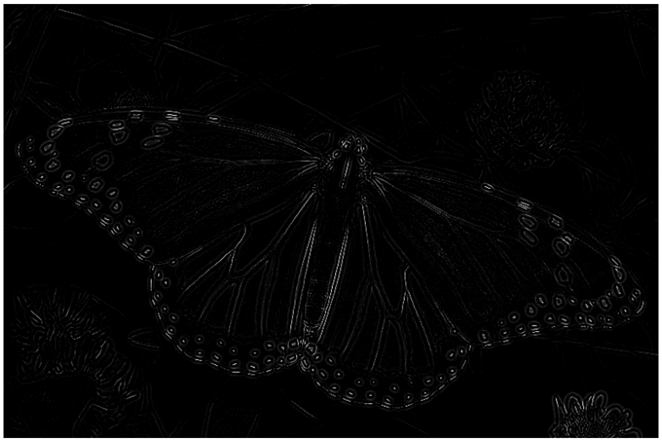
$$\text{Gradient} = \sqrt{I_x^2 + I_y^2}$$



```
def gradient(I, patch_size):
   patch_x = [[1,0,-1],[2,0,-2],[1,0,-1]]
   patch_y = [[1, 2, 1], [0, 0, 0], [-1, -2, -1]]
   Imag = np.zeros([size_i[0] - 2, size_i[1]-2])
   for i in range(size_i[0] - 2):
      for j in range(size_i[1]-2):
         output = np.zeros(patch_size)
         for k in range(patch_size[0]):
            for l in range(patch_size[1]):
               output[k,l] = I[k+i, l+j]
         temp1 = np.sum(patch_x*output) / 8
         temp2 = np.sum(patch_y*output) / 8
         Imag[i, j] = np.sqrt(temp1**2 + temp2**2)
```

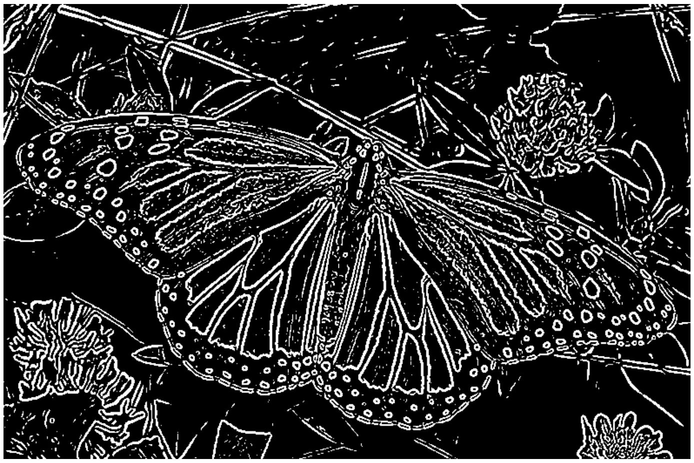
Step 3 Laplacian operator is used as it is a second derivative operator for better extraction of edges.

Laplacian kernel: [[0,1,0],[1,-4,1],[0,1,0]]



```
def laplace(I, patch_size):
   patch = [[0,1,0],[1,-4,1],[0,1,0]]
   I1 = np.zeros([size_i[0] - 2, size_i[1]-2])
   for i in range(size_i[0] - 2):
      for j in range(size_i[1]-2):
         output = np.zeros(patch_size)
         for k in range(patch_size[0]):
            for l in range(patch_size[1]):
               output[k,l] = I[k+i, l+j]
         temp = np.sum(patch*output)
         I1[i, j] = temp
```

Step 4 To make the image brighter and sharper, hysteresis (double threshold) is used where current pixel intensity is calculated using the neighboring pixel values.



```
def hysteresis(I, t1, t2):
   ht = np.where(I > t1)
   lt = np.where(I < t2)
   I[ht] = 255
   I[lt] = 0
   for i in range(1, size_i[0] - 1):
      for j in range(1, size_i[1] - 1):
         if(I[i, j] > t1 and I[i,j] < t2):
            temp = max(I[i+1, j], I[i-1, j], I[i, j+1], I[i, j-1])
            if(temp == 255):
               I[i, j] = 255
            else:
               I[i,j] = 0
    return I
```