Bit Plane Slicing

In Bit-plane slicing, we divide the image into bit planes. This is done by first converting the pixel values in the binary form and then dividing it into bit planes

In [7]:
```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
```

In [8]:
```
# reading image and converting to gray scale
img = Image.open('../images/tiger.jpg').convert('L')
# display image
img
```

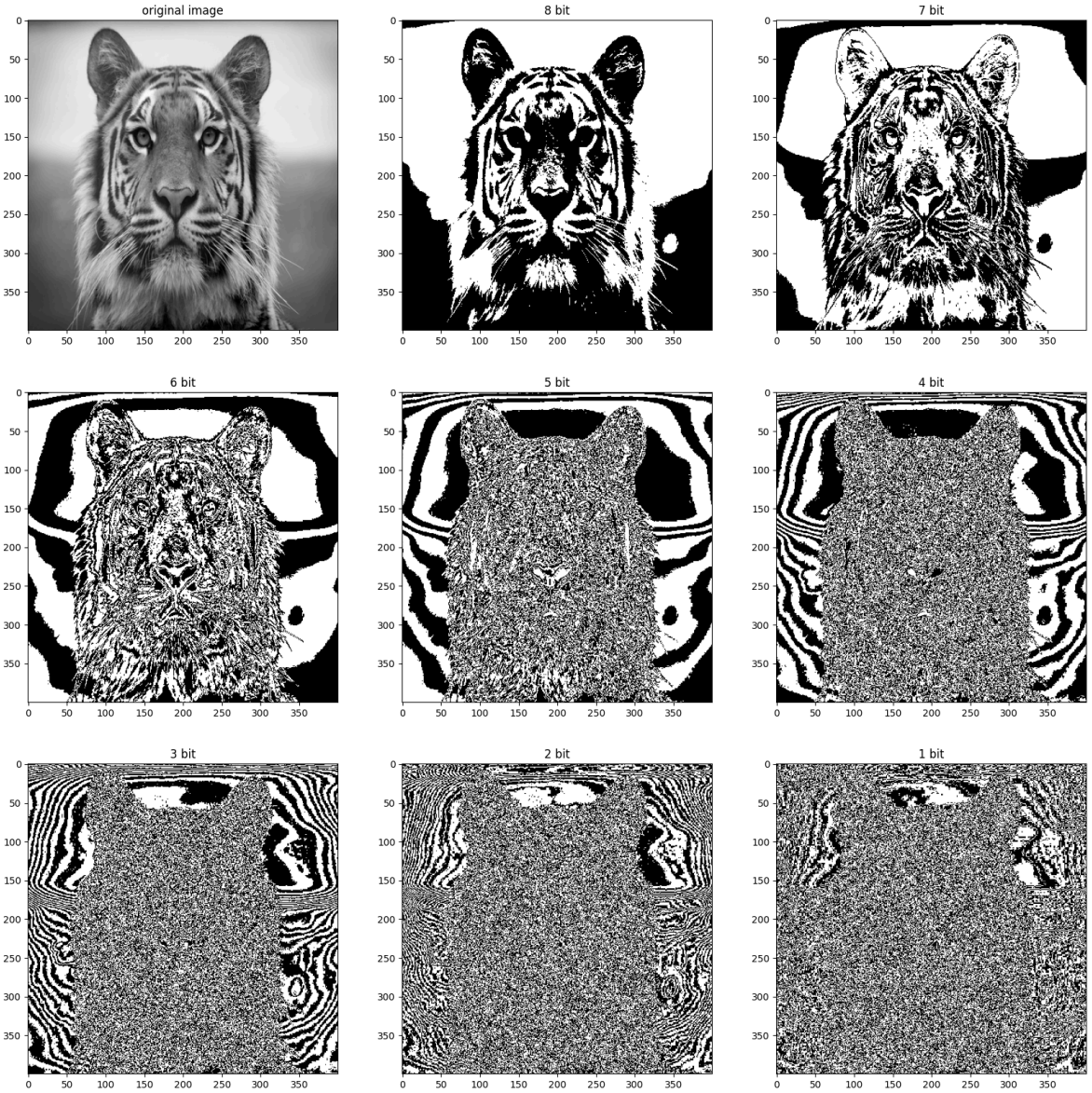
Out[8]:

In [9]:
```
def bit_slicing_function(input_image):
  img = input_image.resize((400,400), Image.Resampling.LANCZOS)
  # convert to numpy array
  numpy_image = np.array(img)

  fig = plt.figure()
  fig.set_figheight(20)
  fig.set_figwidth(20)

  fig.add_subplot(3,3,1)
  plt.imshow(img, cmap='gray')
  plt.title('original image')

  #Iterate over each pixel and change pixel value to binary using
np.binary_repr() and store it in a list.
  lst = []
  for i in range(numpy_image.shape[0]):
```

```
    for j in range(numpy_image.shape[1]):
      lst.append(np.binary_repr(numpy_image[i][j] ,width=8)) # width = no. of
bits

  # We have a list of strings where each string represents binary pixel value.
To extract bit planes we need to iterate over the strings and store the
characters corresponding to bit planes into lists.
  # Multiply with 2^(n-1) and reshape to reconstruct the bit image.
  eight_bit_img = (np.array([int(i[0]) for i in lst],dtype = np.uint8) *
128).reshape(400,400)
  eight_bit_image = Image.fromarray(eight_bit_img)

  fig.add_subplot(3,3,2)
  plt.imshow(eight_bit_image, cmap='gray')
  plt.title('8 bit')


  seven_bit_img = (np.array([int(i[1]) for i in lst],dtype = np.uint8) *
64).reshape(400,400)
  seven_bit_image = Image.fromarray(seven_bit_img)

  fig.add_subplot(3,3,3)
  plt.imshow(seven_bit_image, cmap='gray')
  plt.title('7 bit')

  six_bit_img = (np.array([int(i[2]) for i in lst],dtype = np.uint8) *
32).reshape(400,400)
  six_bit_img = Image.fromarray(six_bit_img)

  fig.add_subplot(3,3,4)
  plt.imshow(six_bit_img, cmap='gray')
  plt.title('6 bit')


  five_bit_img = (np.array([int(i[3]) for i in lst],dtype = np.uint8) *
16).reshape(400,400)
  five_bit_img = Image.fromarray(five_bit_img)

  fig.add_subplot(3,3,5)
  plt.imshow(five_bit_img, cmap='gray')
  plt.title('5 bit')

  four_bit_img = (np.array([int(i[4]) for i in lst],dtype = np.uint8) *
8).reshape(400,400)
  four_bit_img = Image.fromarray(four_bit_img)

  fig.add_subplot(3,3,6)
  plt.imshow(four_bit_img, cmap='gray')
  plt.title('4 bit')


  three_bit_img = (np.array([int(i[5]) for i in lst],dtype = np.uint8) *
4).reshape(400,400)
  three_bit_img = Image.fromarray(three_bit_img)

  fig.add_subplot(3,3,7)
  plt.imshow(three_bit_img, cmap='gray')
  plt.title('3 bit')

  two_bit_img = (np.array([int(i[6]) for i in lst],dtype = np.uint8) *
```

```
2).reshape(400,400)
  two_bit_img = Image.fromarray(two_bit_img)

  fig.add_subplot(3,3,8)
  plt.imshow(two_bit_img, cmap='gray')
  plt.title('2 bit')

  one_bit_img = (np.array([int(i[7]) for i in lst],dtype = np.uint8) *
1).reshape(400,400)
  one_bit_img = Image.fromarray(one_bit_img)

  fig.add_subplot(3,3,9)
  plt.imshow(one_bit_img, cmap='gray')
  plt.title('1 bit')
```

In [10]:
```
bit_slicing_function(img)
```



Gray Level Slicing

It is used to highlight a specific rangeof intensities in an image that might be of interest.

Two common approaches:

- Set all the pixel values with a range of interest to one value (white) and all others to another value (black) produces a binary image.
- Brighten(or darken) pixel values in a range of interest and leave all others unchanged.

It is useful highlighting features in an image.

In [ ]:
```
#import libraries
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
```

In [ ]:
```
#read the image
# reading image and converting to gray scale
img = Image.open('../images/tiger.jpg').convert('L')
# display image
img
```

Out[ ]:

In [ ]:
```
img = img.resize((200,200), Image.Resampling.LANCZOS)
# convert to numpy array
numpy_image = np.array(img)
```

In [ ]:
```
numpy_image.shape
row = numpy_image.shape[0]
```

```
column = numpy_image.shape[1]
```

Set all the pixel values with a range of interest to one value (white) and all others to another value (black) produces a binary image.

In [ ]:
```
new_array = np.zeros(shape=(row,column))
```

In [ ]:
```
for i in range(row):
  for j in range(column):
    new_array[i][j] = numpy_image[i][j]
```

In [ ]:
```
new_array
```

Out[ ]:
```
array([[ 86.,  86.,  88., ..., 138., 138., 136.],
       [ 93.,  93.,  94., ..., 148., 147., 146.],
       [ 98.,  99., 100., ..., 158., 156., 155.],
       ...,
       [ 49.,  49.,  50., ...,  72.,  71.,  71.],
       [ 49.,  49.,  49., ...,  73.,  73.,  72.],
       [ 48.,  49.,  49., ...,  72.,  72.,  72.]], shape=(200, 200))
```

In [ ]:
```
numpy_image
```

Out[ ]:
```
array([[ 86,  86,  88, ..., 138, 138, 136],
       [ 93,  93,  94, ..., 148, 147, 146],
       [ 98,  99, 100, ..., 158, 156, 155],
       ...,
       [ 49,  49,  50, ...,  72,  71,  71],
       [ 49,  49,  49, ...,  73,  73,  72],
       [ 48,  49,  49, ...,  72,  72,  72]], shape=(200, 200), dtype=uint8)
```

In [ ]:
```
for i in range(row):
  for j in range(column):
    if((numpy_image[i][j]>100)&(numpy_image[i][j]<150)):
      new_array[i][j] = 255
    else:
      new_array[i][j] = 0
```

In [ ]:
```
new_array
```

Out[ ]:
```
array([[  0.,   0.,   0., ..., 255., 255., 255.],
       [  0.,   0.,   0., ..., 255., 255., 255.],
       [  0.,   0.,   0., ...,   0.,   0.,   0.],
       ...,
       [  0.,   0.,   0., ...,   0.,   0.,   0.],
       [  0.,   0.,   0., ...,   0.,   0.,   0.],
       [  0.,   0.,   0., ...,   0.,   0.,   0.]], shape=(200, 200))
```

In [ ]:
```
gray_level_slicing_image = Image.fromarray(new_array)
gray_level_slicing_image = gray_level_slicing_image.convert("L")
gray_level_slicing_image
```

Out[ ]:

Brighten(or darken) pixel values in a range of interest and leave all others unchanged.

In [ ]:
```
new_array2 = np.zeros(shape=(row,column))
```

In [ ]:
```
for i in range(row):
  for j in range(column):
    if((numpy_image[i][j]>100)&(numpy_image[i][j]<150)):
      new_array2[i][j] = 0
    else:
      new_array2[i][j] = numpy_image[i][j]
```

In [ ]:
```
gray_level_slicing_image2 = Image.fromarray(new_array2)
gray_level_slicing_image2 = gray_level_slicing_image2.convert("L")
gray_level_slicing_image2
```

Out[ ]:

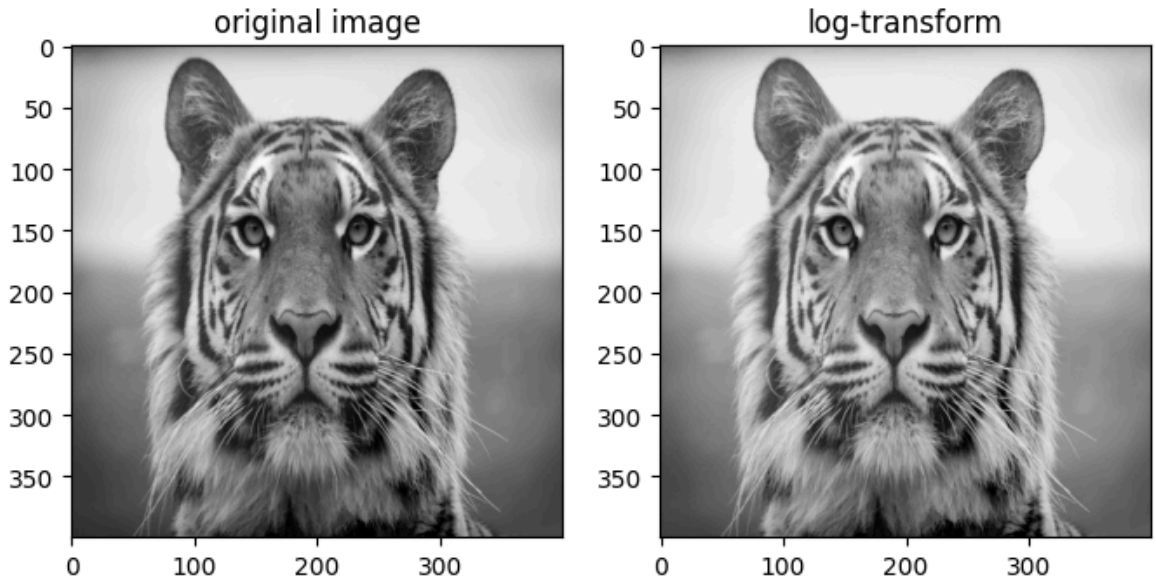
Log Transform

In [1]:
```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
```

In [2]:
```
def log_transform(input_image):
  # resizing image
  img = input_image.resize((400,400), Image.Resampling.LANCZOS)
  # convert to numpy array
  numpy_image = np.array(img)
  numpy_image = numpy_image/255
  numpy_image = numpy_image + 1
  numpy_image = np.log(numpy_image)
  print(type(numpy_image))
  numpy_image = numpy_image * 255
  numpy_image = np.around(numpy_image,decimals=0)
  log_image = Image.fromarray(numpy_image)
  log_image = log_image.convert("L")

    #plotting input and output images
  # set up side-by-side image display
  fig = plt.figure()
  fig.set_figheight(6)
  fig.set_figwidth(8)

  fig.add_subplot(1,2,1)
  plt.imshow(img, cmap='gray')
  plt.title('original image')

  fig.add_subplot(1,2,2)
  plt.imshow(log_image, cmap='gray')
  plt.title('log-transform')

  return log_image
```

In [3]:
```
# reading image and converting to gray scale
img = Image.open("../images/tiger.jpg").convert('L')
# display image
a = log_transform(img)
```

```
<class 'numpy.ndarray'>
```


original image
log-transform
0
50
100
150
200
250
300
350
0
100
200
300

Logic Operators

In [1]:
```
#import libraries
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
```

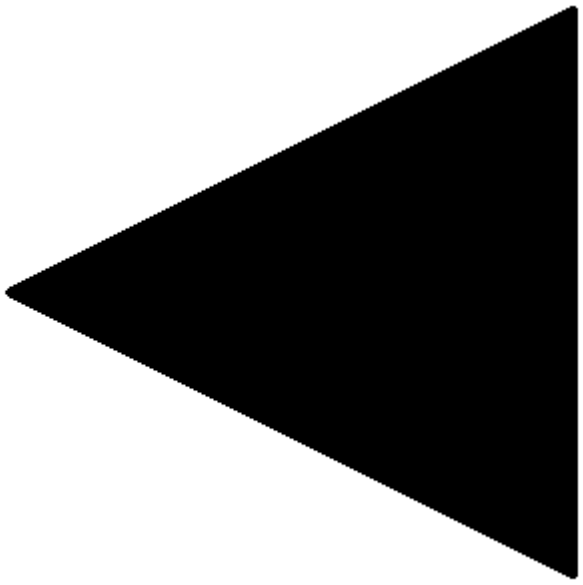
In [2]:
```
#read the image
# reading image and converting to gray scale
img = Image.open("../images/rectangle.png").convert('L')
# display image
img
```

Out[2]:

In [3]:
```
img = img.resize((300,300), Image.Resampling.LANCZOS)
# convert to numpy array
numpy_image = np.array(img)
```

In [4]:
```
#read the image
# reading image and converting to gray scale
img2 = Image.open("../images/triangle.png").convert('L')
# display image
img2
```

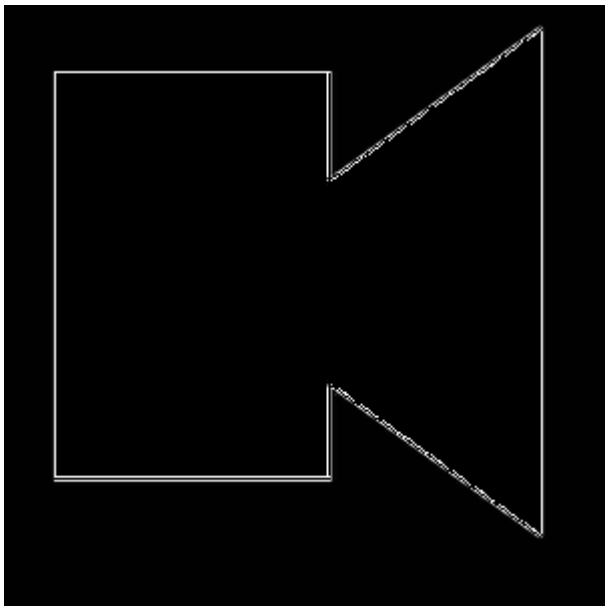
Out[4]:



In [5]:
```
img2 = img2.resize((300,300), Image.Resampling.LANCZOS)
# convert to numpy array
numpy_image2 = np.array(img2)
```

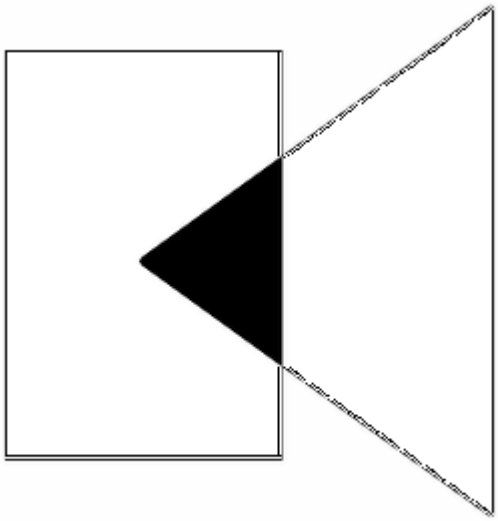
In [6]:
```
# Logic AND
aa = numpy_image*numpy_image2
aa = Image.fromarray(aa)
aa = aa.convert("L")
aa
```

Out[6]:



In [7]:
```
# Logic OR
aa = numpy_image+numpy_image2
aa = Image.fromarray(aa)
aa = aa.convert("L")
aa
```

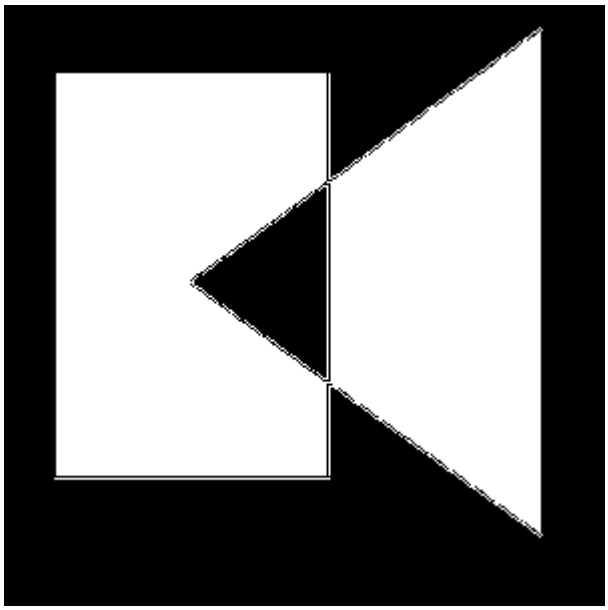
Out[7]:



In [8]:

```
#XOR
aa = np.logical_xor(numpy_image, numpy_image2)
aa = Image.fromarray(aa)
aa = aa.convert("L")
aa
```

Out[8]:



__Negative Images__

Negative images are useful for enhancing white or grey detail embedded in dark regions

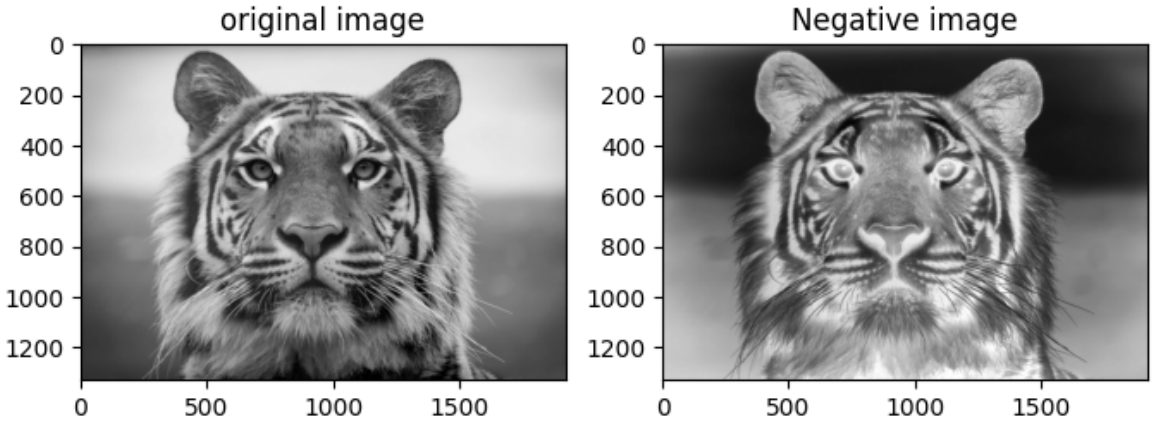
S = max_intensity - r

In [4]:
```
#import libraries
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
```

In [5]:
```
def Negative_Image(image):
  # image = image.resize((400,400), Image.Resampling.LANCZOS)
  # convert to numpy array
  numpy_image = np.array(image)

  row = numpy_image.shape[0]
  column = numpy_image.shape[1]

  new_array = np.zeros(shape=(row,column))

  for i in range(row):
    for j in range(column):
      new_array[i][j] = 255 - numpy_image[i][j]

  negative_image = Image.fromarray(new_array)
  negative_image = negative_image.convert("L")

  return negative_image
```

In [6]:
```
# reading image and converting to gray scale
image = Image.open('../images/tiger.jpg').convert('L')
# calling negative function
negative_image = Negative_Image(image)

#displaying the images
fig = plt.figure()
fig.set_figheight(8)
fig.set_figwidth(8)

fig.add_subplot(1,2,1)
plt.imshow(image, cmap='gray')
plt.title('original image')

fig.add_subplot(1,2,2)
plt.imshow(negative_image, cmap='gray')
plt.title('Negative image')
```

Out[6]:
```
Text(0.5, 1.0, 'Negative image')
```


original image
Negative image
0
200
400
600
800
1000
1200
0
500
1000
1500

__Power Function__

Power law transformations have the following form

$S = C r ^ {\gamma}$

S = output pixel

r = input pixel

C = constant

Map a narrow range of dark input values into a wider range of output values or vice versa Varying γ gives a whole family of curves

In [1]:
```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
```

In [2]:
```
def power_function(input_image):
  img = input_image.resize((400,400), Image.Resampling.LANCZOS)



  # convert to numpy array
  numpy_image = np.array(img)
  numpy_image = numpy_image/255

  #transforming for different values of gamma

  #plotting input and output images
  # set up side-by-side image display
  fig = plt.figure()
  fig.set_figheight(20)
  fig.set_figwidth(20)

  fig.add_subplot(4,4,1)
  plt.imshow(img, cmap='gray')
  plt.title('original image')

  gamma = 0.1
  for i in range(15):
    #print(i*gamma)
    numpy_image_a = np.power(numpy_image,gamma*(i+1))
    numpy_image_a = numpy_image_a * 255
    numpy_image_a = np.around(numpy_image_a,decimals=0)
    power_image = Image.fromarray(numpy_image_a)
    power_image = power_image.convert("L")
    fig.add_subplot(4,4,i+2)
    plt.imshow(power_image, cmap='gray')
    plt.title('gamma = '+ str(round(i*gamma,1)))


  return power_image
```

In [3]:

```
# reading image and converting to gray scale
img = Image.open('../images/tiger.jpg').convert('L')
# Calling the power function
a = power_function(img)
```

__Thresholding__

S = 1 if r>=threshold

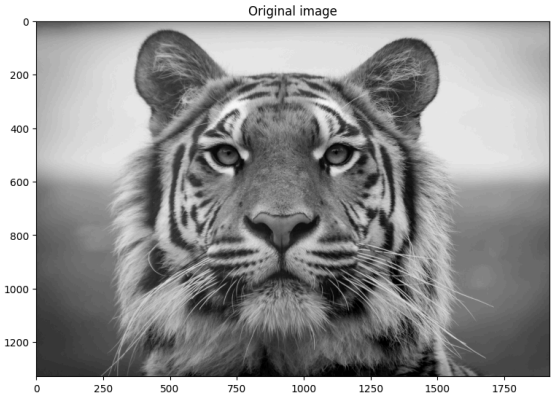
S = 0 if r< threshold

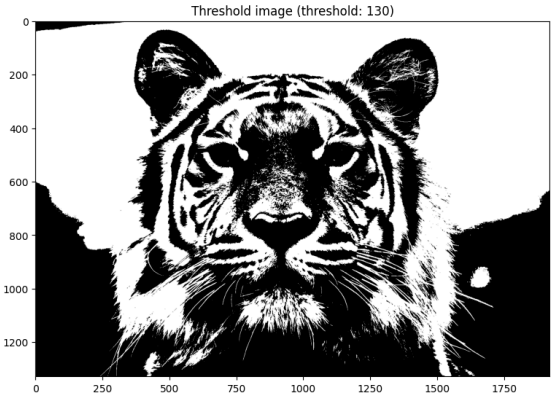
In [1]:
```
#import libraries
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
```

In [2]:
```
def Threshold_Image(image):
  threshold = int(input("Enter Value of threshold: "))
  # image = image.resize((400,400), Image.Resampling.LANCZOS)
  # convert to numpy array
  numpy_image = np.array(image)

  row = numpy_image.shape[0]
  column = numpy_image.shape[1]

  new_array = np.zeros(shape=(row,column))

  for i in range(row):
    for j in range(column):
      if(numpy_image[i][j]>=threshold):
        new_array[i][j] = 255
      else:
        new_array[i][j] = 0

  #converting array back to image
  threshold_image = Image.fromarray(new_array)
  threshold_image = threshold_image.convert("L")

  return threshold_image, threshold
```

In [3]:
```
# reading image and converting to gray scale
image = Image.open('../images/tiger.jpg').convert('L')
# calling negative function
threshold_image, threshold_value = Threshold_Image(image)


#displaying the images
fig = plt.figure()
fig.set_figheight(20)
fig.set_figwidth(20)

fig.add_subplot(1,2,1)
plt.imshow(image, cmap='gray')
plt.title('Original image')

fig.add_subplot(1,2,2)
plt.imshow(threshold_image, cmap='gray')
plt.title(f'Threshold image (threshold: {threshold_value})')
```

Out[3]:
```
Text(0.5, 1.0, 'Threshold image (threshold: 130)')
```


Original image
0
200
400
600
800
1000
1200
0
250
500
750
1000
1250
1500
1750


Threshold image (threshold: 130)
0
200
400
600
800
1000
1200
0
250
500
750
1000
1250
1500
1750

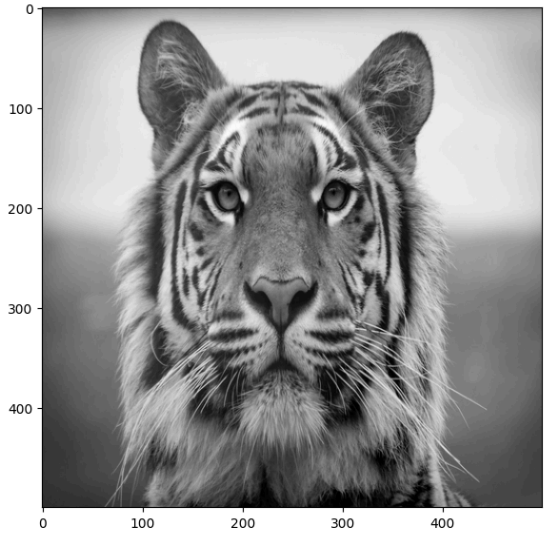
Contrast Stretching

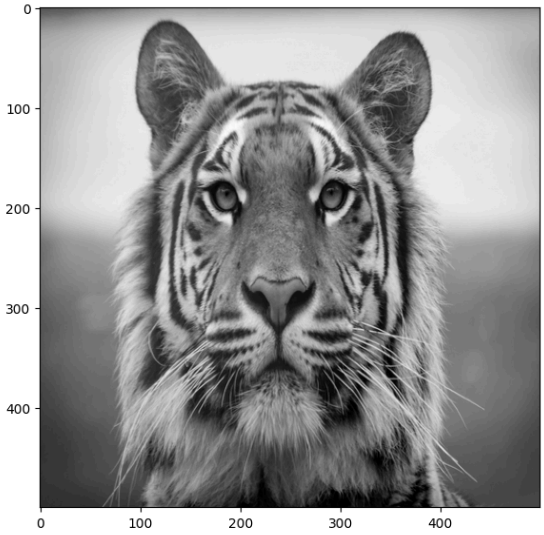
In [1]:
```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
```

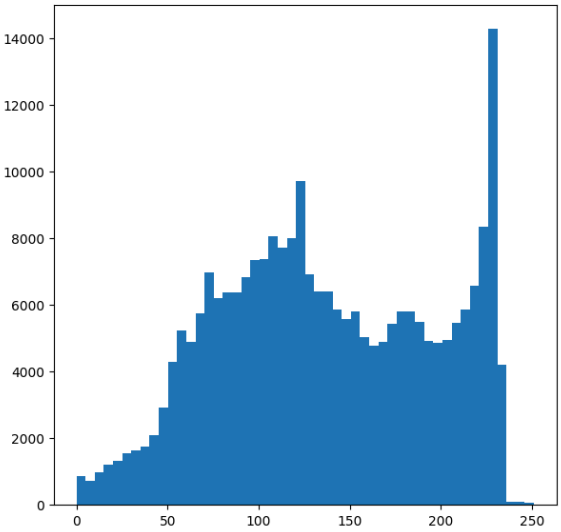
In [2]:
```
def contrast_stretching(input_image):
  img = input_image.resize((500,500), Image.Resampling.LANCZOS)
  #converitng it to array
  img_array = np.asarray(img)
  min = np.amin(img_array)
  print(min)
  max = np.amax(img_array)
  print(max)
  range = max-min
  row = 400
  column = 400

  array_b = np.zeros((row,column))
  array_b = (((img_array - min)/range)*255)
  min = np.amin(array_b)
  print(min)
  max = np.amax(array_b)
  print(max)
  final_image = Image.fromarray(array_b)
  final_image= final_image.convert("L")


    # set up side-by-side image display
  fig = plt.figure()
  fig.set_figheight(15)
  fig.set_figwidth(15)

  fig.add_subplot(2,2,1)
  plt.imshow(img, cmap='gray')

#Plotting Histogram
  flat = img_array.flatten()
  fig.add_subplot(2,2,3)
  plt.hist(flat, bins=50)

  fig.add_subplot(2,2,2)
  plt.imshow(final_image, cmap='gray')

#Plotting Histogram
  flat2 = array_b.flatten()
  fig.add_subplot(2,2,4)
  plt.hist(flat2, bins=50)

  return final_image
```

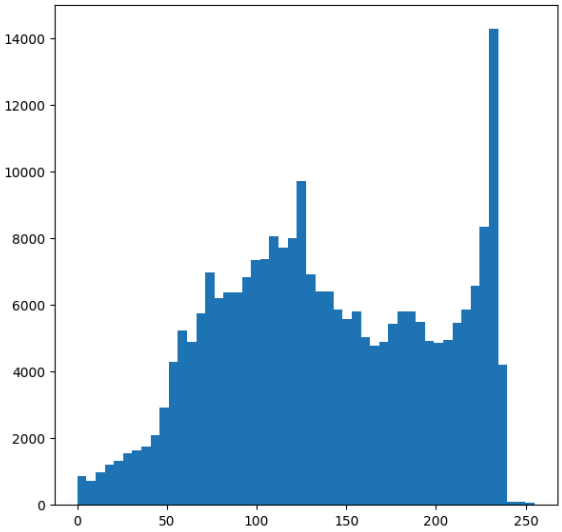
In [3]:
```
# reading image and converting to gray scale
input_image = Image.open("../images/tiger.jpg").convert('L')
a = contrast_stretching(input_image)
```

0
251
0.0
255.0


0
100
200
300
400
0
100
200
300
400


0
100
200
300
400
0
100
200
300
400


14000
12000
10000
8000
6000
4000
2000
0
0
50
100
150
200
250


14000
12000
10000
8000
6000
4000
2000
0
0
50
100
150
200
250

__Histogram Equalization__

The histogram of an image shows us the distribution of grey levels in the image.

Histogram equalization is the process for increasing the contrast in an image by spreading the histogram out to be approximately uniformly distributed.

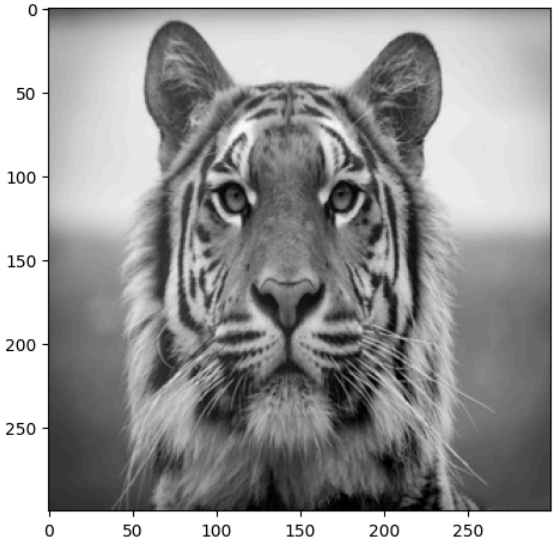
In [1]:
```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
```

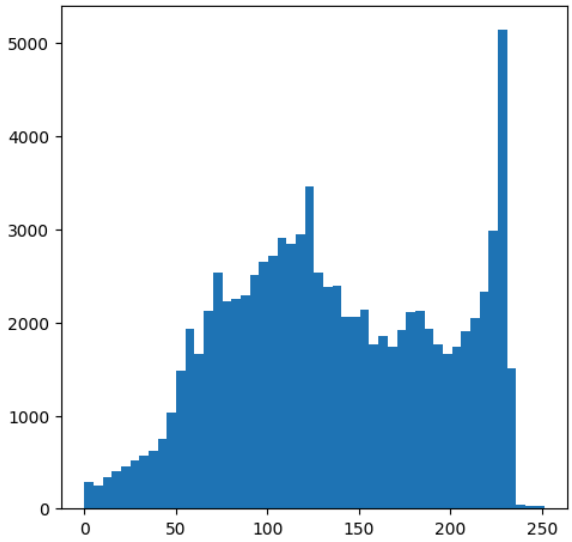
In [2]:
```
def histogram_equalization(input_image):
  # resizing image
  img = input_image.resize((300,300), Image.Resampling.LANCZOS)
  # convert our image into a numpy array
  img = np.asarray(img)

  # put pixels in a 1D array by flattening out img array
  flat = img.flatten()

  # show the histogram
  #plt.hist(flat, bins=256)

  histogram = np.zeros(256)
  # loop through pixels and sum up counts of pixels
  for pixel in flat:
    histogram[pixel] += 1

  #plt.plot(histogram)

  a = iter(histogram)
  b = [next(a)]
  for i in a:
    b.append(b[-1] + i)

  cs = np.array(b)

  # numerator & denomenator
  nj = (cs - cs.min()) * 255
  N = cs.max() - cs.min()

  # re-normalize the cumsum
  cs = nj / N

  # cast it back to uint8 since we can't use floating point values in images
  cs = cs.astype('uint8')

  # get the value from cumulative sum for every index in flat, and set that as
img_new
  img_new = cs[flat]

  # put array back into original shape since we flattened it
  img_new = np.reshape(img_new, img.shape)

  # set up side-by-side image display
```

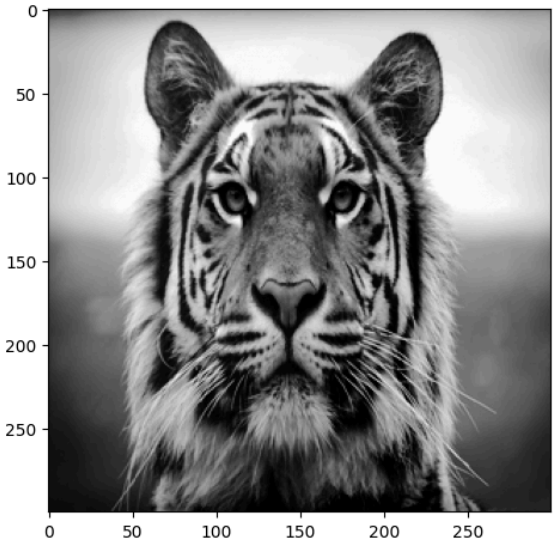
```
  fig = plt.figure()
  fig.set_figheight(12)
  fig.set_figwidth(12)

  fig.add_subplot(2,2,1)
  plt.imshow(img, cmap='gray')

  #histogram of initial image
  fig.add_subplot(2,2,2)
  plt.hist(flat, bins=50)

  # display the new image
  fig.add_subplot(2,2,3)
  plt.imshow(img_new, cmap='gray')

  #histogram of final image
  fig.add_subplot(2,2,4)
  plt.hist(cs, bins=50)

  plt.show(block=True)
```

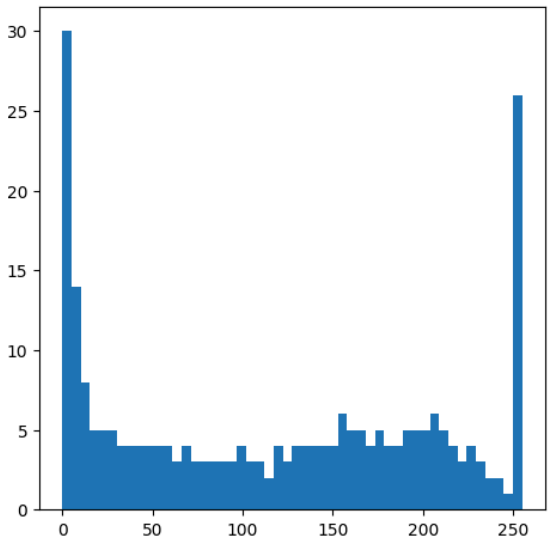
In [3]:

```
# reading image and converting to gray scale
img = Image.open('../images/tiger.jpg').convert('L')
#Calling the histogram equalization function
histogram_equalization(img)
```


0
50
100
150
200
250
0
50
100
150
200
250


5000
4000
3000
2000
1000
0
0
50
100
150
200
250


0
50
100
150
200
250
0
50
100
150
200
250


30
25
20
15
10
5
0
0
50
100
150
200
250

__High Boost__

High-boost = (A-1)Original - High-pass

In [1]:
```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
```

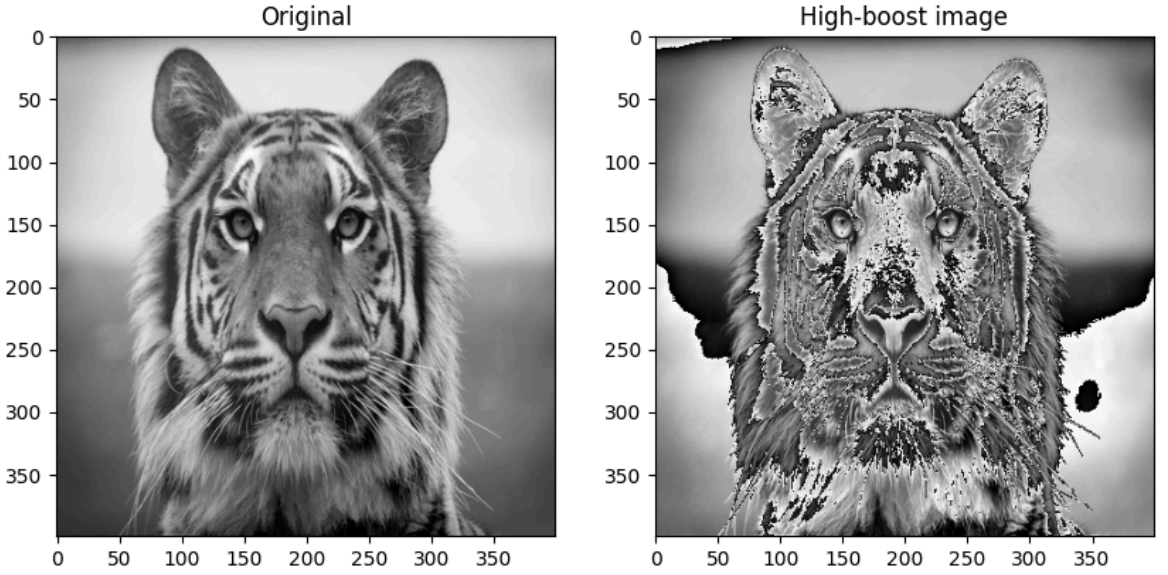
In [2]:
```
def sharp_filter(input_image):
  img = input_image.resize((400,400), Image.Resampling.LANCZOS)

  # convert to numpy array
  numpy_image = np.array(img)
  # array for padding
  array_b = np.zeros((402,402))

  # to pad initial array with zeros in all side
  array_b[1:401,1:401] = numpy_image

  #defining filter
  filter_array = np.array([[-1/9,-1/9,-1/9],
                           [-1/9,8/9,-1/9],
                           [-1/9,-1/9,-1/9]])

  #creating empty list
  lst = []

  for i in range(400):
    for j in range(400):
      #extracting part of array equal to filter size
      array_c = array_b[i:(3+i),j:(3+j)]

     #applying filter
      array_mul = np.multiply(filter_array,array_c)
      array_sum = np.sum(array_mul)

      # putting calculated value in list
      lst.append(array_sum)

  # resizing lst to shape of original array
  final_array = np.resize(lst,(400,400))

  final_image = Image.fromarray(final_array)
  final_image= final_image.convert("L")

  return final_image
```

In [3]:
```
def high_boost(input_image):
  high_pass = sharp_filter(input_image)
  img = input_image.resize((400,400), Image.Resampling.LANCZOS)
  # convert to numpy array
  numpy_image = np.array(img)

  A = 3
  high_boost = (A-1)*numpy_image + high_pass
```

```
  final_image = Image.fromarray(high_boost)
  final_image= final_image.convert("L")

  fig = plt.figure()
  fig.set_figheight(10)
  fig.set_figwidth(10)

  #plotting original image
  fig.add_subplot(1,2,1)
  plt.imshow(img, cmap='gray')
  plt.title('Original')

  #plotting filtered image
  fig.add_subplot(1,2,2)
  plt.imshow(final_image, cmap='gray')
  plt.title('High-boost image')

  return final_image
```

In [4]:

```
# reading image and converting to gray scale
img = Image.open('../images/tiger.jpg').convert('L')
# Calling smooth function
a = high_boost(img)
```



Median Filter

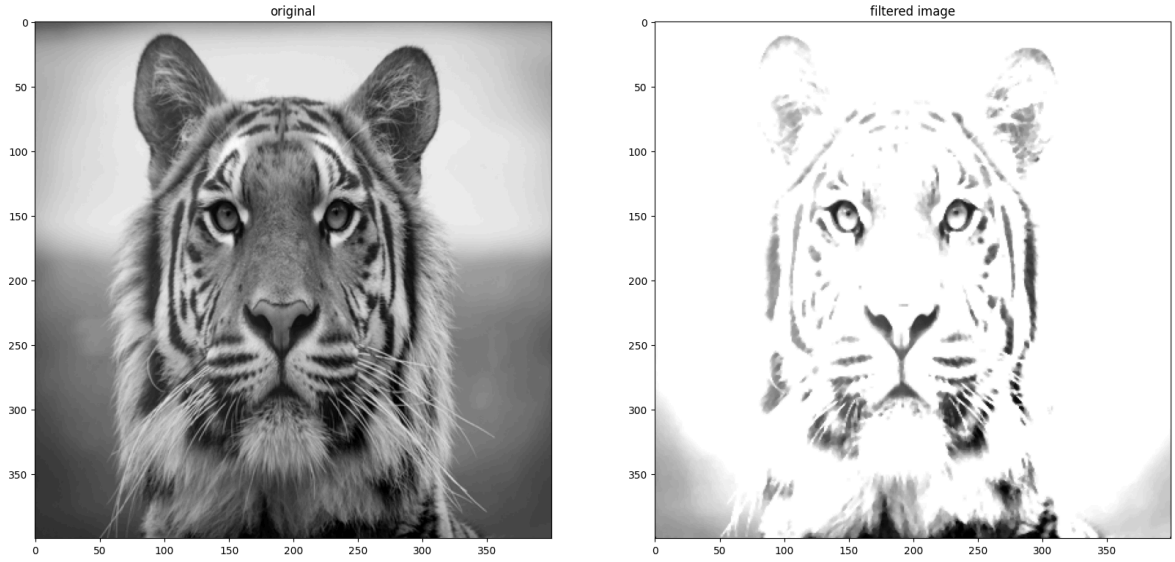
In [1]:
```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
```

In [2]:
```
def median_filter(input_image):
  img = input_image.resize((400,400), Image.Resampling.LANCZOS)

  fig = plt.figure()
  fig.set_figheight(20)
  fig.set_figwidth(20)

  #plotting original image
  fig.add_subplot(1,2,1)
  plt.imshow(img, cmap='gray')
  plt.title('original')

  # convert to numpy array
  numpy_image = np.array(img)
  # array for padding
  array_b = np.zeros((402,402))

  # to pad initial array with zeros in all side
  array_b[1:401,1:401] = numpy_image

  #defining filter
  #filter_array = np.array([[1,1,1],
   #                        [1,1,1],
    #                       [1,1,1]])

  filter_array = np.array([[3,3,3],
                           [3,3,3],
                           [3,3,3]])

  #creating empty list
  lst = []

  for i in range(400):
    for j in range(400):
      #extracting part of array equal to filter size
      array_c = array_b[i:(3+i),j:(3+j)]

     #applying filter
      array_mul = np.multiply(filter_array,array_c)
      array_sum = np.median(array_mul)

      # putting calculated value in list
      lst.append(array_sum)

  # resizing lst to shape of original array
  final_array = np.resize(lst,(400,400))

  final_image = Image.fromarray(final_array)
  final_image= final_image.convert("L")
```

```
    #plotting filtered image
  fig.add_subplot(1,2,2)
  plt.imshow(final_image, cmap='gray')
  plt.title('filtered image')
```

In [3]:

```
# reading image and converting to gray scale
img = Image.open('../images/tiger.jpg').convert('L')
# Call filter function
median_filter(img)
```



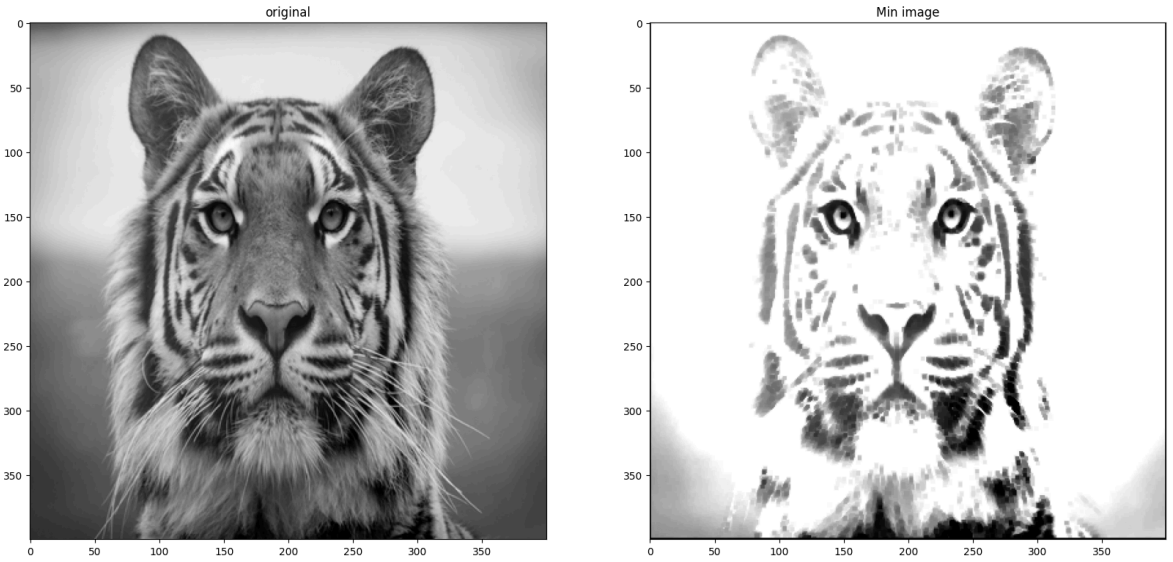
Min Max Filter

In [1]:
```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
```

In [2]:
```
def min_filter(input_image):
  img = input_image.resize((400,400), Image.Resampling.LANCZOS)

  fig = plt.figure()
  fig.set_figheight(20)
  fig.set_figwidth(20)

  #plotting original image
  fig.add_subplot(1,2,1)
  plt.imshow(img, cmap='gray')
  plt.title('original')


  # convert to numpy array
  numpy_image = np.array(img)
  # array for padding
  array_b = np.zeros((402,402))

  # to pad initial array with zeros in all side
  array_b[1:401,1:401] = numpy_image

  #defining filter
  #filter_array = np.array([[1,1,1],
   #                        [1,1,1],
    #                       [1,1,1]])

  filter_array = np.array([[3,3,3],
                           [3,3,3],
                           [3,3,3]])

  #creating empty list
  lst = []

  for i in range(400):
    for j in range(400):
      #extracting part of array equal to filter size
      array_c = array_b[i:(3+i),j:(3+j)]

     #applying filter
      array_mul = np.multiply(filter_array,array_c)
      array_sum = np.min(array_mul)

      # putting calculated value in list
      lst.append(array_sum)

  # resizing lst to shape of original array
  final_array = np.resize(lst,(400,400))

  final_image = Image.fromarray(final_array)
  final_image= final_image.convert("L")
```

```
  #plotting filtered image
  fig.add_subplot(1,2,2)
  plt.imshow(final_image, cmap='gray')
  plt.title('Min image')
```

In [3]:
```
# reading image and converting to gray scale
img = Image.open('../images/tiger.jpg').convert('L')
# Call filter function
min_filter(img)
```

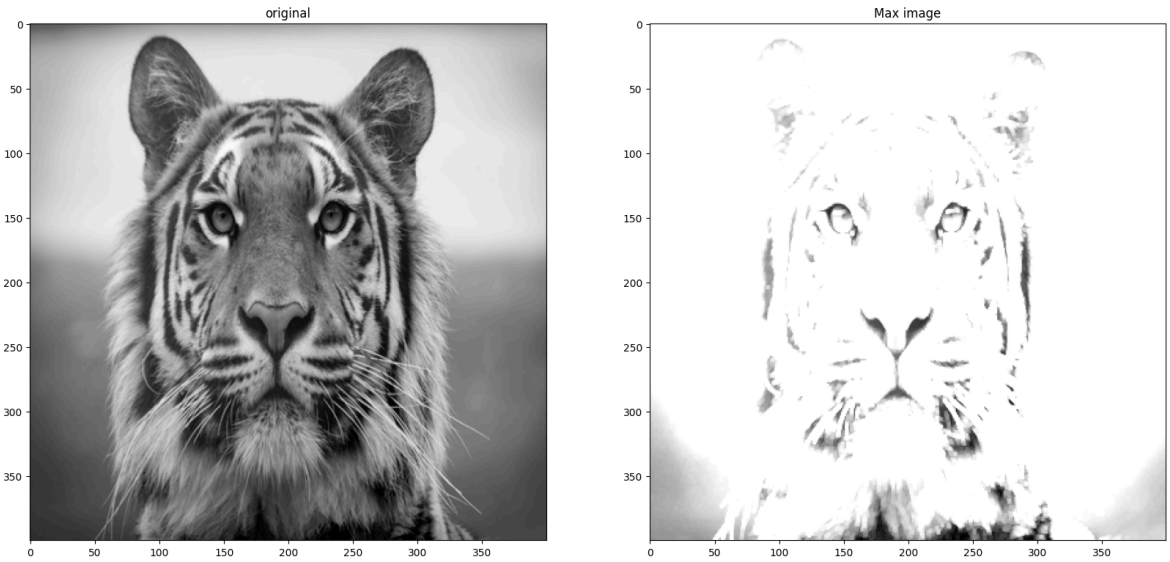


In [4]:
```
def max_filter(input_image):
  img = input_image.resize((400,400), Image.Resampling.LANCZOS)

  fig = plt.figure()
  fig.set_figheight(20)
  fig.set_figwidth(20)

  #plotting original image
  fig.add_subplot(1,2,1)
  plt.imshow(img, cmap='gray')
  plt.title('original')


  # convert to numpy array
  numpy_image = np.array(img)
  # array for padding
  array_b = np.zeros((402,402))

  # to pad initial array with zeros in all side
  array_b[1:401,1:401] = numpy_image

  #defining filter
  #filter_array = np.array([[1,1,1],
   #                        [1,1,1],
    #                       [1,1,1]])

  filter_array = np.array([[3,3,3],
                           [3,3,3],
                           [3,3,3]])
```

```
  #creating empty list
  lst = []

  for i in range(400):
    for j in range(400):
      #extracting part of array equal to filter size
      array_c = array_b[i:(3+i),j:(3+j)]

     #applying filter
      array_mul = np.multiply(filter_array,array_c)
      array_sum = np.max(array_mul)

      # putting calculated value in list
      lst.append(array_sum)

  # resizing lst to shape of original array
  final_array = np.resize(lst,(400,400))

  final_image = Image.fromarray(final_array)
  final_image= final_image.convert("L")

    #plotting filtered image
  fig.add_subplot(1,2,2)
  plt.imshow(final_image, cmap='gray')
  plt.title('Max image')
```

In [5]:

```
# reading image and converting to gray scale
img = Image.open('../images/tiger.jpg').convert('L')
# Call max filter function
max_filter(img)
```



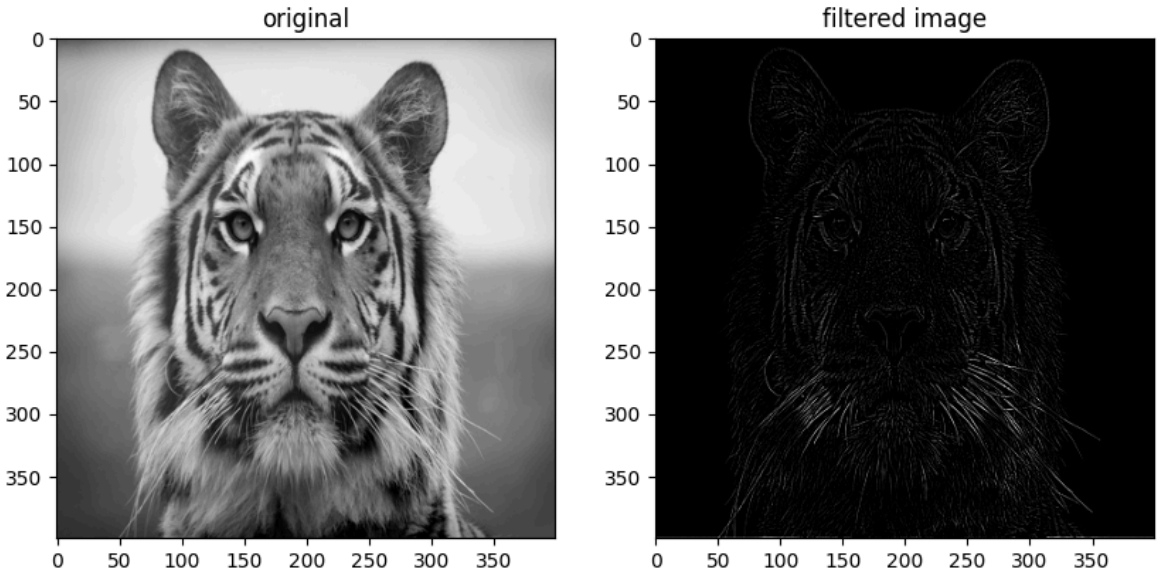
Sharpening Filter

In [1]:
```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
```

In [2]:
```
def sharp_filter(input_image):
  img = input_image.resize((400,400), Image.Resampling.LANCZOS)

  fig = plt.figure()
  fig.set_figheight(10)
  fig.set_figwidth(10)

  #plotting original image
  fig.add_subplot(1,2,1)
  plt.imshow(img, cmap='gray')
  plt.title('original')


  # convert to numpy array
  numpy_image = np.array(img)
  # array for padding
  array_b = np.zeros((402,402))

  # to pad initial array with zeros in all side
  array_b[1:401,1:401] = numpy_image

  #defining filter
  filter_array = np.array([[-1/9,-1/9,-1/9],
                           [-1/9,8/9,-1/9],
                           [-1/9,-1/9,-1/9]])

  #creating empty list
  lst = []

  for i in range(400):
    for j in range(400):
      #extracting part of array equal to filter size
      array_c = array_b[i:(3+i),j:(3+j)]

     #applying filter
      array_mul = np.multiply(filter_array,array_c)
      array_sum = np.sum(array_mul)

      # putting calculated value in list
      lst.append(array_sum)

  # resizing lst to shape of original array
  final_array = np.resize(lst,(400,400))

  final_image = Image.fromarray(final_array)
  final_image= final_image.convert("L")

    #plotting filtered image
  fig.add_subplot(1,2,2)
```

```
  plt.imshow(final_image, cmap='gray')
  plt.title('filtered image')
```

In [3]:
```
# reading image and converting to gray scale
img = Image.open('../images/tiger.jpg').convert('L')
# Sharpen the image
sharp_filter(img)
```



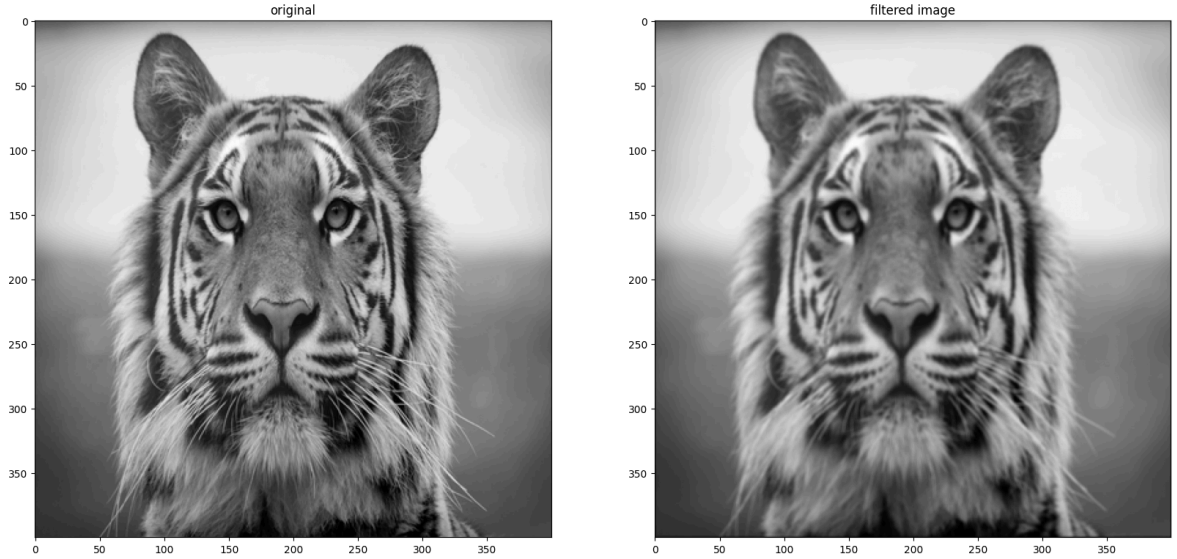
Smoothing Filter

In [1]:
```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
```

In [2]:
```
def smooth_filter(input_image):
  img = input_image.resize((400,400), Image.Resampling.LANCZOS)

  fig = plt.figure()
  fig.set_figheight(20)
  fig.set_figwidth(20)

  #plotting original image
  fig.add_subplot(1,2,1)
  plt.imshow(img, cmap='gray')
  plt.title('original')


  # convert to numpy array
  numpy_image = np.array(img)
  # array for padding
  array_b = np.zeros((402,402))

  # to pad initial array with zeros in all side
  array_b[1:401,1:401] = numpy_image

  #defining filter
  filter_array = np.array([[1/9,1/9,1/9],
                           [1/9,1/9,1/9],
                           [1/9,1/9,1/9]])

  #creating empty list
  lst = []

  for i in range(400):
    for j in range(400):
      #extracting part of array equal to filter size
      array_c = array_b[i:(3+i),j:(3+j)]

     #applying filter
      array_mul = np.multiply(filter_array,array_c)
      array_sum = np.sum(array_mul)

      # putting calculated value in list
      lst.append(array_sum)

  # resizing lst to shape of original array
  final_array = np.resize(lst,(400,400))

  final_image = Image.fromarray(final_array)
  final_image= final_image.convert("L")

    #plotting filtered image
  fig.add_subplot(1,2,2)
```

```
  plt.imshow(final_image, cmap='gray')
  plt.title('filtered image')
```

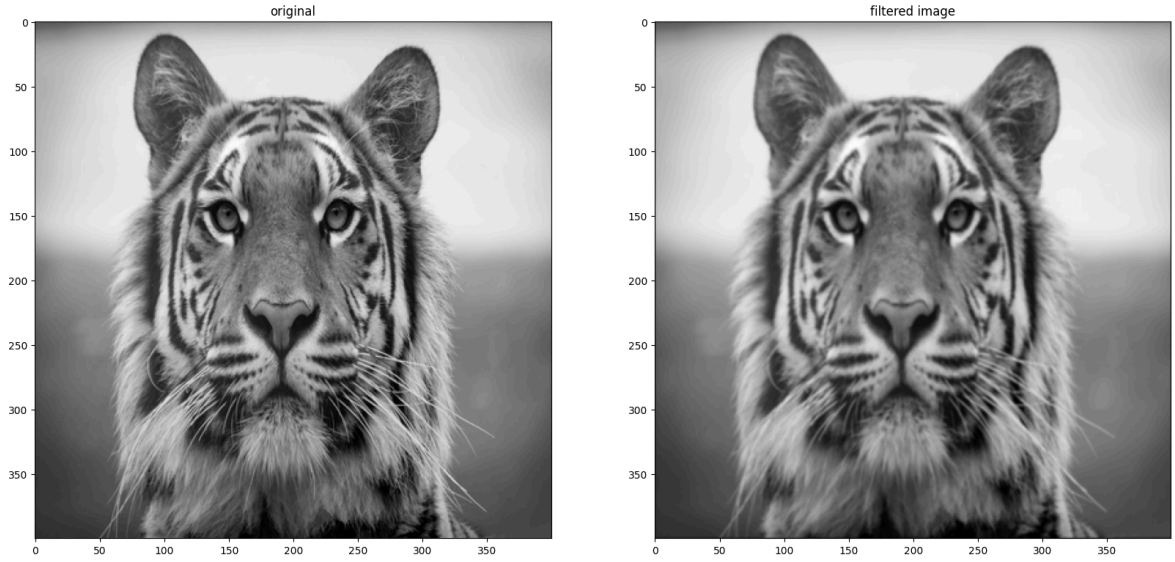
In [3]:
```
# reading image and converting to gray scale
img = Image.open('../images/tiger.jpg').convert('L')
# Calling smooth function
smooth_filter(img)
```



In [4]:
```
def weighted_smooth_filter(input_image):
  img = input_image.resize((400,400), Image.Resampling.LANCZOS)

  fig = plt.figure()
  fig.set_figheight(20)
  fig.set_figwidth(20)

  #plotting original image
  fig.add_subplot(1,2,1)
  plt.imshow(img, cmap='gray')
  plt.title('original')

  # convert to numpy array
  numpy_image = np.array(img)
  # array for padding
  array_b = np.zeros((402,402))

  # to pad initial array with zeros in all side
  array_b[1:401,1:401] = numpy_image

  #defining filter
  filter_array = np.array([[1/16,2/16,1/16],
                           [2/16,4/16,2/16],
                           [1/16,2/16,1/16]])
  #creating empty list
  lst = []

  for i in range(400):
    for j in range(400):
      #extracting part of array equal to filter size
      array_c = array_b[i:(3+i),j:(3+j)]

     #applying filter
      array_mul = np.multiply(filter_array,array_c)
```

```
        array_sum = np.sum(array_mul)

        # putting calculated value in list
        lst.append(array_sum)

    # resizing lst to shape of original array
    final_array = np.resize(lst,(400,400))

    final_image = Image.fromarray(final_array)
    final_image= final_image.convert("L")

      #plotting filtered image
    fig.add_subplot(1,2,2)
    plt.imshow(final_image, cmap='gray')
    plt.title('filtered image')
```

In [5]:
```
weighted_smooth_filter(img)
```



1st Derivative Filter

Prewitt Operator

$$ \begin{bmatrix} -1 & -1 & -1\\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} $$$$ \begin{bmatrix} -1 & 0 & 1\\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} $$

In [1]:
```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
```

In [2]:
```
def Prewitt_Operator_horizontal(input_image):
  img = input_image.resize((400,400), Image.Resampling.LANCZOS)

  # convert to numpy array
  numpy_image = np.array(img)
  # array for padding
  array_b = np.zeros((402,402))

  # to pad initial array with zeros in all side
  array_b[1:401,1:401] = numpy_image

  #defining filter
  filter_array = np.array([[-1,-1,-1],
                           [0,0,0],
                           [1,1,1]])

  #creating empty list
  lst = []

  for i in range(400):
    for j in range(400):
      #extracting part of array equal to filter size
      array_c = array_b[i:(3+i),j:(3+j)]

     #applying filter
      array_mul = np.multiply(filter_array,array_c)
      array_sum = np.sum(array_mul)

      # putting calculated value in list
      lst.append(array_sum)

  # resizing lst to shape of original array
  final_array = np.resize(lst,(400,400))

  final_image = Image.fromarray(final_array)
  final_image= final_image.convert("L")

  return final_image
```

In [3]:
```
def Prewitt_Operator_vertical(input_image):
  img = input_image.resize((400,400), Image.Resampling.LANCZOS)

  # convert to numpy array
  numpy_image = np.array(img)
```

```
  # array for padding
  array_b = np.zeros((402,402))

  # to pad initial array with zeros in all side
  array_b[1:401,1:401] = numpy_image

  #defining filter
  filter_array = np.array([[-1,0,1],
                           [-1,0,1],
                           [-1,0,1]])

  #creating empty list
  lst = []

  for i in range(400):
    for j in range(400):
      #extracting part of array equal to filter size
      array_c = array_b[i:(3+i),j:(3+j)]

     #applying filter
      array_mul = np.multiply(filter_array,array_c)
      array_sum = np.sum(array_mul)

      # putting calculated value in list
      lst.append(array_sum)

  # resizing lst to shape of original array
  final_array = np.resize(lst,(400,400))

  final_image = Image.fromarray(final_array)
  final_image= final_image.convert("L")

  return final_image
```

__Sobel Operator__

$$ \begin{bmatrix} -1 & -2 & -1\\ 0 & 2 & 0 \\ 1 & 2 & 1 \end{bmatrix} $$$$ \begin{bmatrix} -1 & 0 & 1\\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} $$

In [4]:
```
def Sobel_Operator_horizontal(input_image):
  img = input_image.resize((400,400), Image.Resampling.LANCZOS)

  # convert to numpy array
  numpy_image = np.array(img)
  # array for padding
  array_b = np.zeros((402,402))

  # to pad initial array with zeros in all side
  array_b[1:401,1:401] = numpy_image

  #defining filter
  filter_array = np.array([[-1,-2,-1],
                           [0,0,0],
                           [1,2,1]])

  #creating empty list
  lst = []

  for i in range(400):
```

```
    for j in range(400):
      #extracting part of array equal to filter size
      array_c = array_b[i:(3+i),j:(3+j)]

     #applying filter
      array_mul = np.multiply(filter_array,array_c)
      array_sum = np.sum(array_mul)

      # putting calculated value in list
      lst.append(array_sum)

  # resizing lst to shape of original array
  final_array = np.resize(lst,(400,400))

  final_image = Image.fromarray(final_array)
  final_image= final_image.convert("L")

  return final_image
```

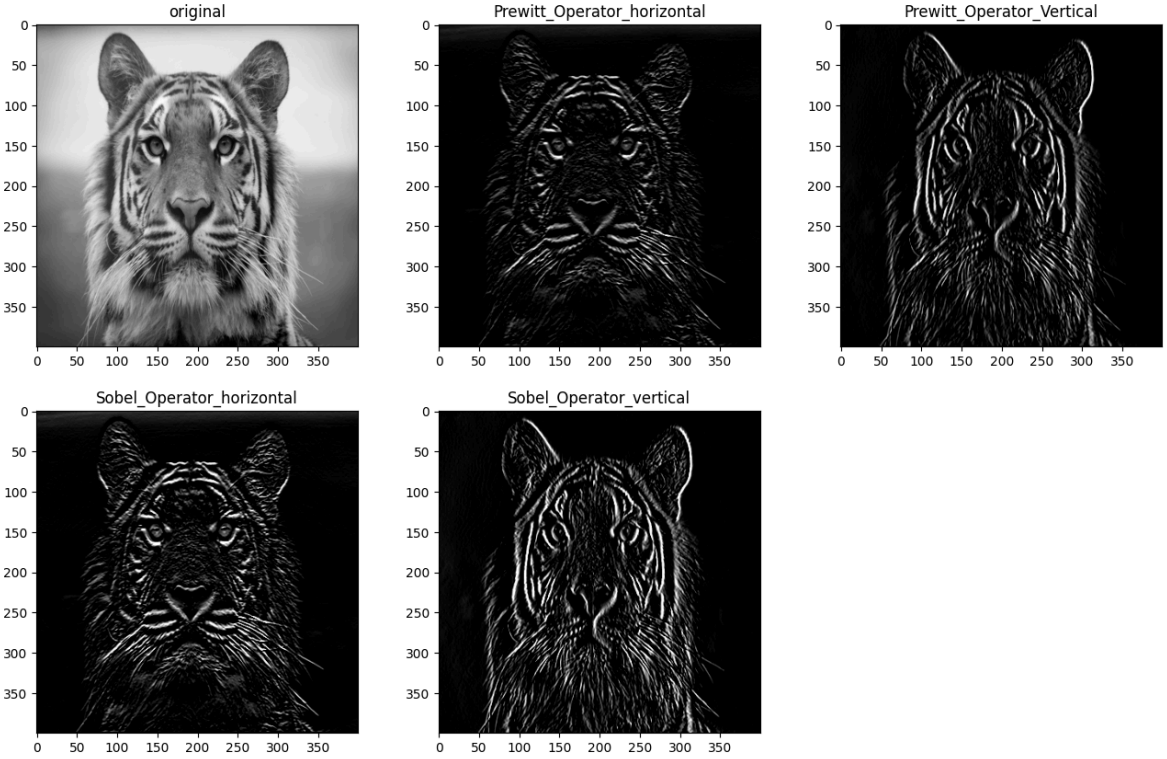
In [5]:

```
def Sobel_Operator_vertical(input_image):
  img = input_image.resize((400,400), Image.Resampling.LANCZOS)

  # convert to numpy array
  numpy_image = np.array(img)
  # array for padding
  array_b = np.zeros((402,402))

  # to pad initial array with zeros in all side
  array_b[1:401,1:401] = numpy_image

  #defining filter
  filter_array = np.array([[-1,0,1],
                           [-2,0,2],
                           [-1,0,1]])

  #creating empty list
  lst = []

  for i in range(400):
    for j in range(400):
      #extracting part of array equal to filter size
      array_c = array_b[i:(3+i),j:(3+j)]

     #applying filter
      array_mul = np.multiply(filter_array,array_c)
      array_sum = np.sum(array_mul)

      # putting calculated value in list
      lst.append(array_sum)

  # resizing lst to shape of original array
  final_array = np.resize(lst,(400,400))

  final_image = Image.fromarray(final_array)
  final_image= final_image.convert("L")

  return final_image
```

In [6]:
```
# reading image and converting to gray scale
img = Image.open('../images/tiger.jpg').convert('L').resize((400,400))
# img = input_image.resize((400,400), Image.Resampling.LANCZOS)
# Calling function
Prewitt_Operator_horizontal_image = Prewitt_Operator_horizontal(img)
Prewitt_Operator_vertical_image = Prewitt_Operator_vertical(img)
Sobel_Operator_horizontal_image = Sobel_Operator_horizontal(img)
Sobel_Operator_vertical_image = Sobel_Operator_vertical(img)
```

In [7]:
```
fig = plt.figure()
fig.set_figheight(10)
fig.set_figwidth(16)

#plotting original image
fig.add_subplot(2,3,1)
plt.imshow(img, cmap='gray')
plt.title('original')

  #plotting filtered image
fig.add_subplot(2,3,2)
plt.imshow(Prewitt_Operator_horizontal_image, cmap='gray')
plt.title('Prewitt_Operator_horizontal')


#plotting filtered image
fig.add_subplot(2,3,3)
plt.imshow(Prewitt_Operator_vertical_image, cmap='gray')
plt.title('Prewitt_Operator_Vertical')

#plotting filtered image
fig.add_subplot(2,3,4)
plt.imshow(Sobel_Operator_horizontal_image, cmap='gray')
plt.title('Sobel_Operator_horizontal')

#plotting filtered image
fig.add_subplot(2,3,5)
plt.imshow(Sobel_Operator_vertical_image, cmap='gray')
plt.title('Sobel_Operator_vertical')
```

Out[7]:
```
Text(0.5, 1.0, 'Sobel_Operator_vertical')
```


original
Prewitt_Operator_horizontal
Prewitt_Operator_Vertical
Sobel_Operator_horizontal
Sobel_Operator_vertical

__Laplacian Filter__

$$ \begin{bmatrix} 0 & 1 & 0\\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} $$

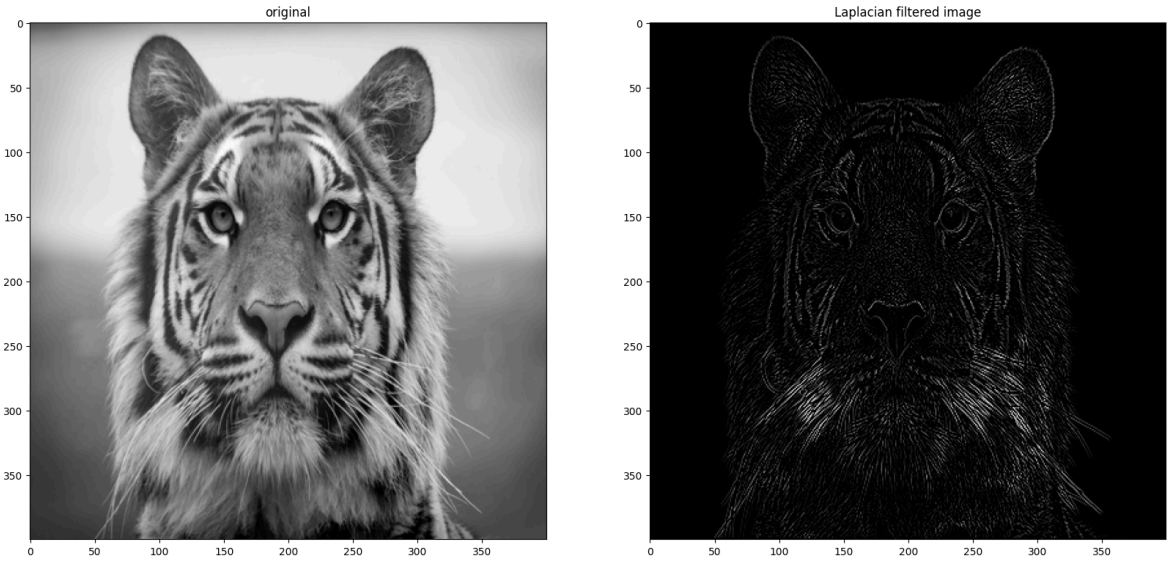
Applying Laplacian filter to an image get a new image that highlights edges and other discontinuties

In [1]:
```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
```

In [2]:
```
def Laplacian_filter(input_image):
  img = input_image.resize((400,400), Image.Resampling.LANCZOS)

  fig = plt.figure()
  fig.set_figheight(20)
  fig.set_figwidth(20)

  #plotting original image
  fig.add_subplot(1,2,1)
  plt.imshow(img, cmap='gray')
  plt.title('original')


  # convert to numpy array
  numpy_image = np.array(img)
  # array for padding
  array_b = np.zeros((402,402))

  # to pad initial array with zeros in all side
  array_b[1:401,1:401] = numpy_image

  #defining filter
  filter_array = np.array([[0,1,0],
                           [1,-4,1],
                           [0,1,0]])

  #creating empty list
  lst = []

  for i in range(400):
    for j in range(400):
      #extracting part of array equal to filter size
      array_c = array_b[i:(3+i),j:(3+j)]

     #applying filter
      array_mul = np.multiply(filter_array,array_c)
      array_sum = np.sum(array_mul)

      # putting calculated value in list
      lst.append(array_sum)

  # resizing lst to shape of original array
  final_array = np.resize(lst,(400,400))

  final_image = Image.fromarray(final_array)
```

```
  final_image= final_image.convert("L")

    #plotting filtered image
  fig.add_subplot(1,2,2)
  plt.imshow(final_image, cmap='gray')
  plt.title('Laplacian filtered image')
```

In [3]:
```
# reading image and converting to gray scale
img = Image.open('../images/tiger.jpg').convert('L')
# Calling function
Laplacian_filter(img)
```



__Variants of Laplace__

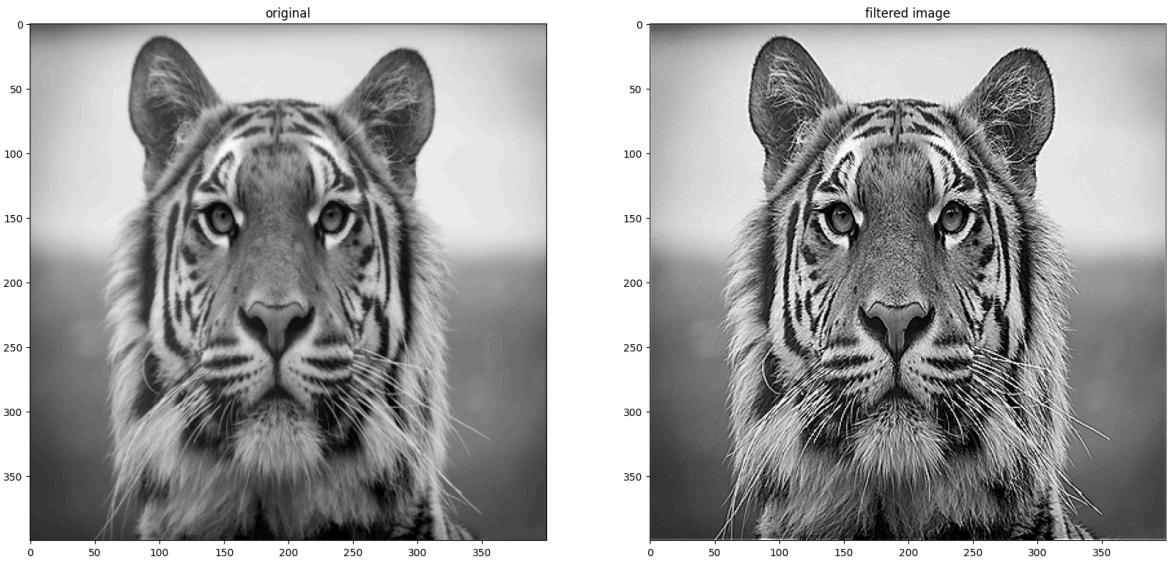
subtract laplace from original image to generate our final sharpened enhanced image

$$ \begin{bmatrix} 0 & -1 & 0\\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} $$

In [4]:
```
def Sharpened_Laplacian_filter(input_image):
  img = input_image.resize((400,400), Image.Resampling.LANCZOS)

  fig = plt.figure()
  fig.set_figheight(20)
  fig.set_figwidth(20)

  #plotting original image
  fig.add_subplot(1,2,1)
  plt.imshow(img, cmap='gray')
  plt.title('original')

  # convert to numpy array
  numpy_image = np.array(img)
  # array for padding
  array_b = np.zeros((402,402))

  # to pad initial array with zeros in all side
  array_b[1:401,1:401] = numpy_image

  #defining filter
  filter_array = np.array([[0,-1,0],
                           [-1,5,-1],
                           [0,-1,0]])
```

```
  #creating empty list
  lst = []

  for i in range(400):
    for j in range(400):
      #extracting part of array equal to filter size
      array_c = array_b[i:(3+i),j:(3+j)]

     #applying filter
      array_mul = np.multiply(filter_array,array_c)
      array_sum = np.sum(array_mul)

      # putting calculated value in list
      lst.append(array_sum)

  # resizing lst to shape of original array
  final_array = np.resize(lst,(400,400))

  final_image = Image.fromarray(final_array)
  final_image= final_image.convert("L")

    #plotting filtered image
  fig.add_subplot(1,2,2)
  plt.imshow(final_image, cmap='gray')
  plt.title('filtered image')
```

In [5]:
```
# reading image and converting to gray scale
img = Image.open('../images/tiger.jpg').convert('L')
# Calling function
Sharpened_Laplacian_filter(img)
```



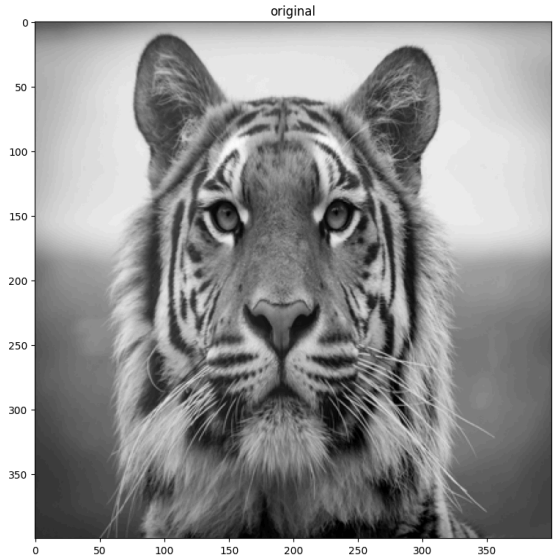
Variant 1

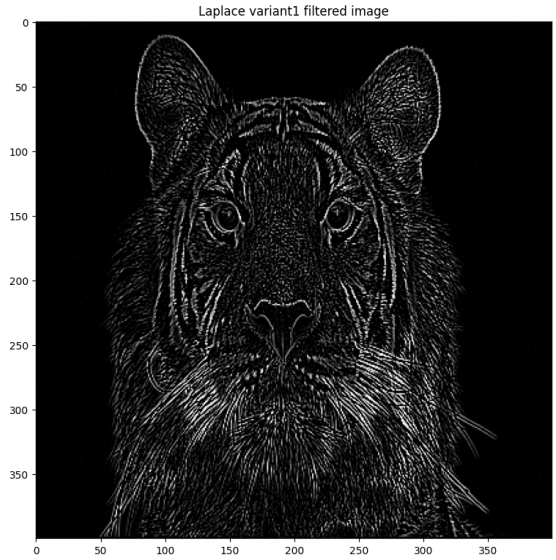
$$ \begin{bmatrix} 1 & 1 & 1\\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} $$

In [6]:
```
def Laplacian_Variant1(input_image):
  img = input_image.resize((400,400), Image.Resampling.LANCZOS)

  fig = plt.figure()
  fig.set_figheight(20)
  fig.set_figwidth(20)
```

```
#plotting original image
fig.add_subplot(1,2,1)
plt.imshow(img, cmap='gray')
plt.title('original')

# convert to numpy array
numpy_image = np.array(img)
# array for padding
array_b = np.zeros((402,402))

# to pad initial array with zeros in all side
array_b[1:401,1:401] = numpy_image

#defining filter
filter_array = np.array([[1,1,1],
                         [1,-8,1],
                         [1,1,1]])

#creating empty list
lst = []

for i in range(400):
  for j in range(400):
    #extracting part of array equal to filter size
    array_c = array_b[i:(3+i),j:(3+j)]

   #applying filter
    array_mul = np.multiply(filter_array,array_c)
    array_sum = np.sum(array_mul)

    # putting calculated value in list
    lst.append(array_sum)

# resizing lst to shape of original array
final_array = np.resize(lst,(400,400))

final_image = Image.fromarray(final_array)
final_image= final_image.convert("L")

  #plotting filtered image
fig.add_subplot(1,2,2)
plt.imshow(final_image, cmap='gray')
plt.title('Laplace variant1 filtered image')
```

In [7]:
```
Laplacian_Variant1(img)
```





Variant 2

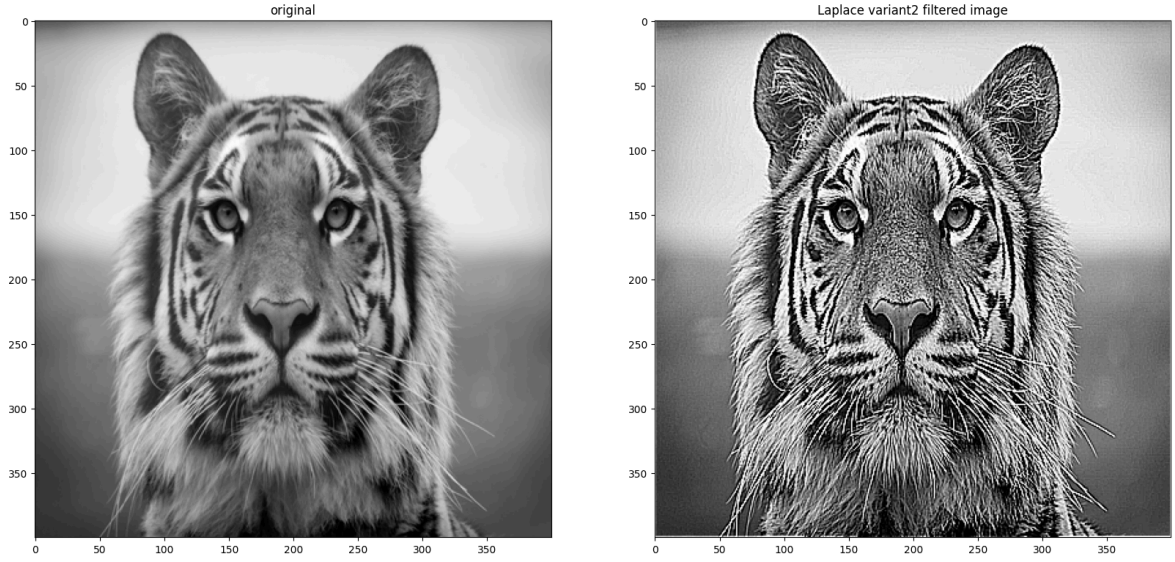
$$ \begin{bmatrix} -1 & -1 & -1\\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix} $$

In [8]:

```
def Laplacian_Variant2(input_image):
  img = input_image.resize((400,400), Image.Resampling.LANCZOS)

  fig = plt.figure()
  fig.set_figheight(20)
  fig.set_figwidth(20)

  #plotting original image
  fig.add_subplot(1,2,1)
  plt.imshow(img, cmap='gray')
  plt.title('original')

  # convert to numpy array
  numpy_image = np.array(img)
  # array for padding
  array_b = np.zeros((402,402))

  # to pad initial array with zeros in all side
  array_b[1:401,1:401] = numpy_image

  #defining filter
  filter_array = np.array([[-1,-1,-1],
                           [-1,9,-1],
                           [-1,-1,-1]])

  #creating empty list
  lst = []

  for i in range(400):
    for j in range(400):
      #extracting part of array equal to filter size
      array_c = array_b[i:(3+i),j:(3+j)]

     #applying filter
      array_mul = np.multiply(filter_array,array_c)
      array_sum = np.sum(array_mul)

      # putting calculated value in list
      lst.append(array_sum)
```

```
# resizing lst to shape of original array
final_array = np.resize(lst,(400,400))

final_image = Image.fromarray(final_array)
final_image= final_image.convert("L")

  #plotting filtered image
fig.add_subplot(1,2,2)
plt.imshow(final_image, cmap='gray')
plt.title('Laplace variant2 filtered image')
```

In [9]:
```
Laplacian_Variant2(img)
```



__Frequency Domain Filtering__

Steps in frequency domain filtering

- Compute the Fourier Transform
- Multiply the result by a filter transform function
- Take the inverse transform to produce the enchanced image

In [1]:
```
from scipy import fftpack
import numpy as np
import imageio.v2 as imageio
from PIL import Image, ImageDraw
import matplotlib.pyplot as plt
```

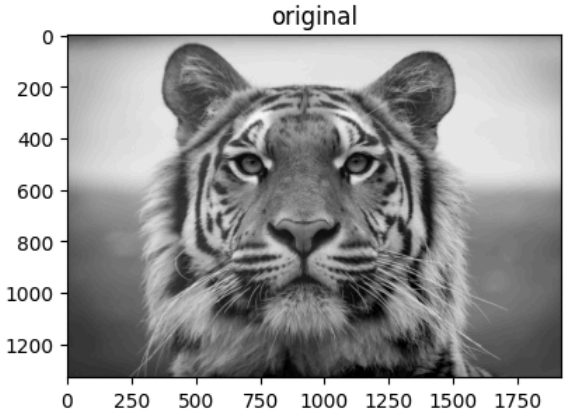
In [2]:
```
def frequency_image(image):
  image_array = np.array(image)
  fft1 = fftpack.fftshift(fftpack.fft2(image_array))
  magnitude_spectrum = 20*np.log(np.abs(fft1))
  freq_image = Image.fromarray(magnitude_spectrum)
  freq_image = freq_image.convert("L")
  return freq_image
```

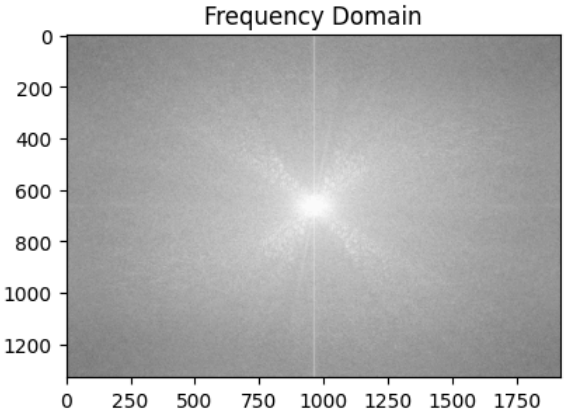
In [3]:
```
def low_pass_filter(image1):
  #convert image to numpy array
  image1_np=np.array(image1)
  #fft of image
  fft1 = fftpack.fftshift(fftpack.fft2(image1_np))
  #Create a low pass filter image
  x,y = image1_np.shape[0],image1_np.shape[1]

  #defining filter
  #size of circle
  e_x,e_y=50,50
  #create a box
  bbox=((x/2)-(e_x/2),(y/2)-(e_y/2),(x/2)+(e_x/2),(y/2)+(e_y/2))
  low_pass=Image.new("L",(image1_np.shape[0],image1_np.shape[1]),color=0)
  draw1=ImageDraw.Draw(low_pass)
  draw1.ellipse(bbox, fill=1)
  low_pass_np=np.array(low_pass)
  low_pass_np = low_pass_np.T
  #end of defining filter

  #multiply both the images
  filtered=np.multiply(fft1,low_pass_np)

  #inverse fft
  ifft2 = np.real(fftpack.ifft2(fftpack.ifftshift(filtered)))
  ifft2 = np.maximum(0, np.minimum(ifft2, 255))
  data = Image.fromarray(ifft2)
  data = data.convert("L")

  return data
```

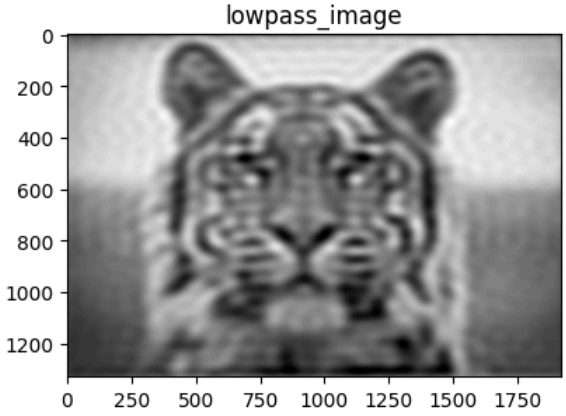
In [4]:
```
def high_pass_filter(image):
  #converting image to array
  image_array = np.array(image)

  #sending image to low pass filter
  lowpass_image = low_pass_filter(image)
  #converting image to array
  lowpass_image_array = np.array(lowpass_image)

  #subtracting lowpass image from original to obtain highpass image
  high_pass_array = image_array - lowpass_image_array

  #array to image
  high_pass_image = Image.fromarray(high_pass_array)
  high_pass_image = high_pass_image.convert("L")

  return high_pass_image
```

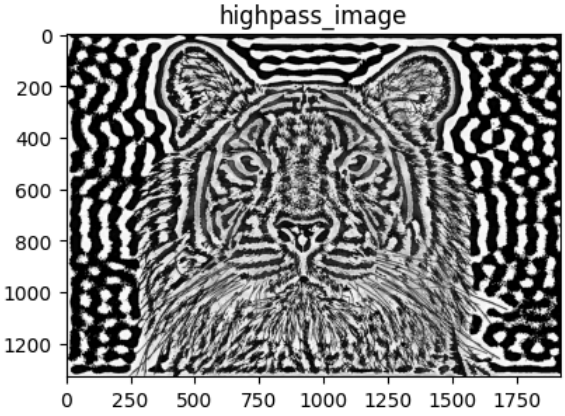
In [5]:
```
image = imageio.imread('../images/tiger.jpg', mode='L')
freq_image = frequency_image(image)
lowpass_image = low_pass_filter(image)
highpass_image = high_pass_filter(image)

fig = plt.figure()
fig.set_figheight(12)
fig.set_figwidth(10)

#plotting original image
fig.add_subplot(2,2,1)
plt.imshow(image, cmap='gray')
plt.title('original')

#plotting the image in frequency domain
fig.add_subplot(2,2,2)
plt.imshow(freq_image, cmap='gray')
plt.title('Frequency Domain')

#plotting lowpass image
fig.add_subplot(2,2,3)
plt.imshow(lowpass_image, cmap='gray')
plt.title('lowpass_image')

#plotting highpass image
fig.add_subplot(2,2,4)
plt.imshow(highpass_image, cmap='gray')
plt.title('highpass_image')
```

Out[5]:
```
Text(0.5, 1.0, 'highpass_image')
```


original
0
200
400
600
800
1000
1200
0
250
500
750
1000
1250
1500
1750


Frequency Domain
0
200
400
600
800
1000
1200
0
250
500
750
1000
1250
1500
1750


lowpass_image
0
200
400
600
800
1000
1200
0
250
500
750
1000
1250
1500
1750


highpass_image
0
200
400
600
800
1000
1200
0
250
500
750
1000
1250
1500
1750