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
```

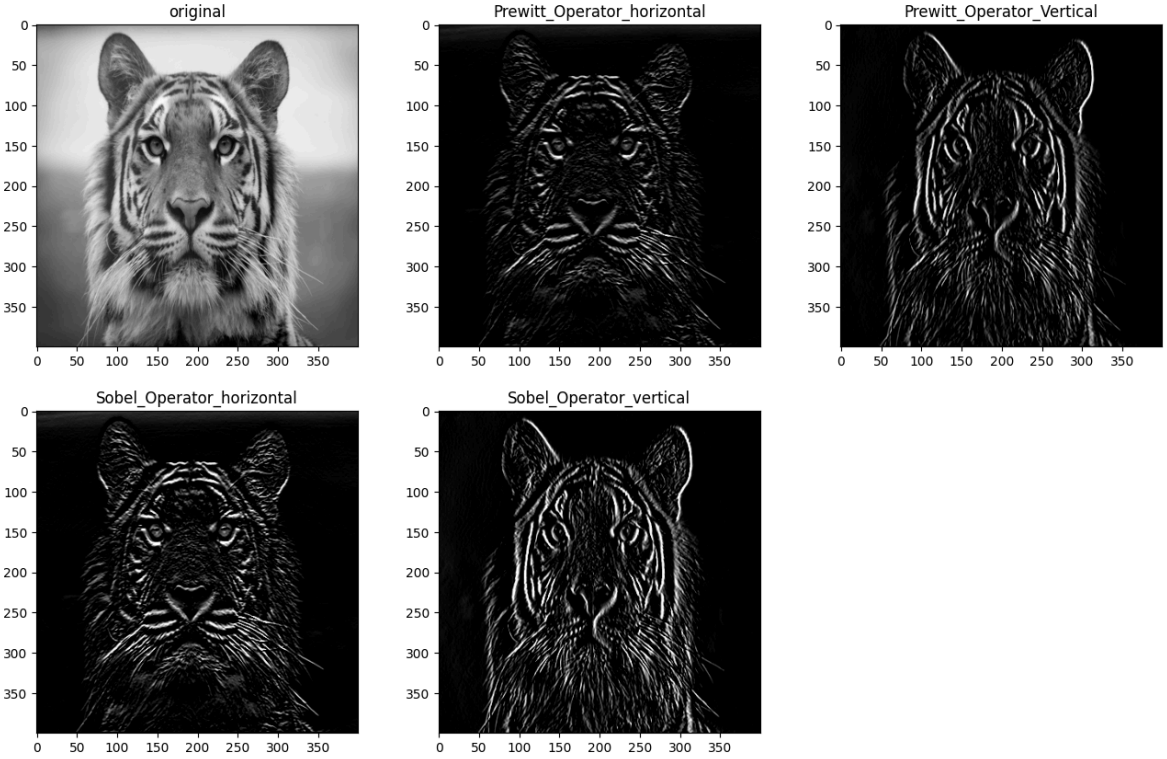
In [5]:

```
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