In [1]:

```


# Implementation of sigmoid function

import numpy as np
def sigmoid(z):
 return 1 / (1 + np.exp(-z))
print("Sigmoid of 4 is:",sigmoid (4))
```

```
Sigmoid of 4 is: 0.9820137900379085
```

In [2]:

```
print("Sigmoid of positive number(5) is:",sigmoid(5))
```

```
Sigmoid of positive number(5) is: 0.9933071490757153
```

In [3]:

```
print("Sigmoid of negative number(-5) is:",sigmoid(-5))
```

```
Sigmoid of negative number(-5) is: 0.0066928509242848554
```

In [4]:

```
print("Difference between Derivative of Sigmoid (5) and (-5) is:", sigmoid(5)*(1- sigmoid(5
```

```
Difference between Derivative of Sigmoid (5) and (-5) is: -1.214306433183765
e-16
```

In [6]:

```
# vanishing gradient

print("Difference between sigmoid of 14 and 15:",sigmoid(15)-sigmoid(14))
```

```
Difference between sigmoid of 14 and 15: 5.256258007735326e-07
```

In [7]:

```
# Implementation of tanh function

def tanh(z):
 return np.tanh(z)
print("tanh of 4 is:",tanh(4))
```

```
tanh of 4 is: 0.999329299739067
```

In [8]:

```
# zero centric

print("tanh of positive number(15) is:",tanh(15))
```

```
tanh of positive number(15) is: 0.9999999999998128
```

In [9]:

```
print("tanh of positive number(-15) is:",tanh(-15))
```

```
tanh of positive number(-15) is: -0.9999999999998128
```

In [10]:

```
# vanishing gradient
```

```
Difference between tanh of 14 and 15: 1.1957101975212936e-12
```

In [12]:

```
# Implementation of ReLU(Rectified Linear Unit) function

def relu(z):
  return max(0, z)
z= 10
```

```
ReLU of 10 is : 10
```

In [13]:

```
# dying neuron( all neurons from (0,-inf)=0, this causes problem in backpropagation)

z= -0.4
print("ReLU of "+str(z)+" is :",relu(z))
z= -50
print("ReLU of "+str(z)+" is :",z * (z > 0))
```

```
ReLU of -0.4 is : 0
ReLU of -50 is : 0
```

In [14]:

```
# Implementation of leaky relu function

def leakyrelu(z):
  return np.maximum(0.01 * z, z)
z= 10
print("ReLU of "+str(z)+" is :",leakyrelu(z))
```

```
ReLU of 10 is : 10.0
```

In [15]:

```
z= -1
print("ReLU of "+str(z)+" is :",leakyrelu(z))
```

```
ReLU of -1 is : -0.01
```

In [16]:

```
# Implementation of Exponential Relu function

def erelu(z,alpha):
    return z if z >= 0 else alpha*(np.exp(z) -1)
print("Exponential ReLu for 10 is :",erelu(10,3))
print("Exponential ReLu for -10 is :",erelu(-10,3))
```

```
Exponential ReLu for 10 is : 10
Exponential ReLu for -10 is : -2.9998638002107123
```

In [21]:

```
#Implementation of filter (of size 3*3 and 5*5) on an image

import cv2
import matplotlib.pyplot as plt


def main():

    path = "D:\\Youtube Code\\Python\\Python OpenCV3\\Python-OpenCV3\\Dataset\\"
    imgpath =  path + "4.2.07.tiff"
    img = cv2.imread(imgpath, 1)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    box = cv2.boxFilter(img, -1, (3, 3))
    blur = cv2.blur(img, (5, 5))

    gaussian = cv2.GaussianBlur(img, (3, 3), 0)

    titles = ['Original Image', 'Box Filter',
              'Blur', 'Gaussian Blur']

    outputs = [img, box, blur, gaussian]


    for i in range(4):
        plt.subplot(2, 2, i+1)
        plt.imshow(outputs[i])
        plt.title(titles[i])
        plt.xticks([])
        plt.yticks([])
    plt.show()
```

In [23]:

```
# Zero padding and stride = 1

def convolve2D(image, kernel, padding=0, strides=1):
    # Cross Correlation
    kernel = np.flipud(np.fliplr(kernel))

    # Gather Shapes of Kernel + Image + Padding
    xKernShape = kernel.shape[0]
    yKernShape = kernel.shape[1]
    xImgShape = image.shape[0]
    yImgShape = image.shape[1]

    # Shape of Output Convolution
    xOutput = int(((xImgShape - xKernShape + 2 * padding) / strides) + 1)
    yOutput = int(((yImgShape - yKernShape + 2 * padding) / strides) + 1)
    output = np.zeros((xOutput, yOutput))

    # Apply Equal Padding to All Sides
    if padding != 0:
        imagePadded = np.zeros((image.shape[0] + padding*2, image.shape[1] + padding*2))
        imagePadded[int(padding):int(-1 * padding), int(padding):int(-1 * padding)] = image
        print(imagePadded)
    else:
        imagePadded = image

    # Iterate through image
    for y in range(image.shape[1]):
        # Exit Convolution
        if y > image.shape[1] - yKernShape:
            break
        # Only Convolve if y has gone down by the specified Strides
        if y % strides == 0:
            for x in range(image.shape[0]):
                # Go to next row once kernel is out of bounds
                if x > image.shape[0] - xKernShape:
                    break
                try:
                    # Only Convolve if x has moved by the specified Strides
                    if x % strides == 0:
                        output[x, y] = (kernel * imagePadded[x: x + xKernShape, y: y + yKer
                except:
                    break

    return output
```

In [ ]:

```
```