

# Assignment - 2

Name - prekshita patil

Reg.No - 20MAI0073

Link: <https://github.com/prekshita19/DL-Assignments> (<https://github.com/prekshita19/DL-Assignments>).

Que 1 :- Apply all activation Function

```
In [1]: #Import all the library use in the activation function
import matplotlib.pyplot as plt
import numpy as np
```

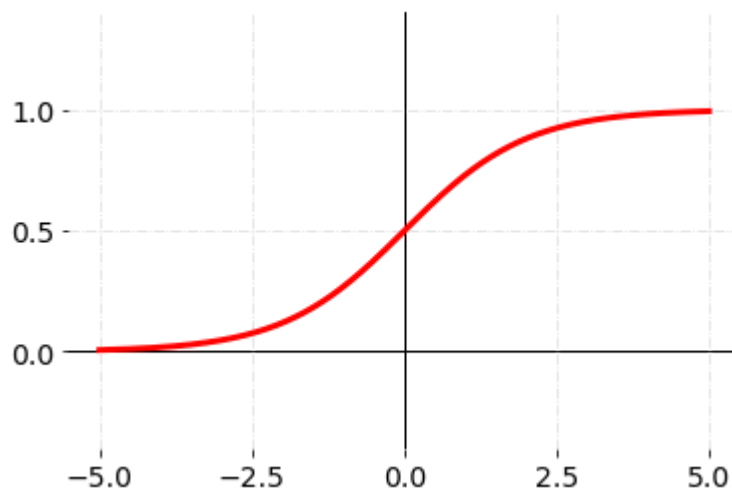
```
In [2]: #Give the range to the X-axis
x = np.arange(-5, 5, 0.01)
```

```
In [3]: def plot(func, yaxis=(-1.4, 1.4)):
    plt.ylim(yaxis)
    plt.locator_params(nbins=5)
    plt.xticks(fontsize = 14)
    plt.yticks(fontsize = 14)
    plt.axhline(lw=1, c='black')
    plt.axvline(lw=1, c='black')
    plt.grid(alpha=0.4, ls='-.')
    plt.box(on=None)
    plt.plot(x, func(x), c='r', lw=3)
```

## Sigmoid

```
In [4]: def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

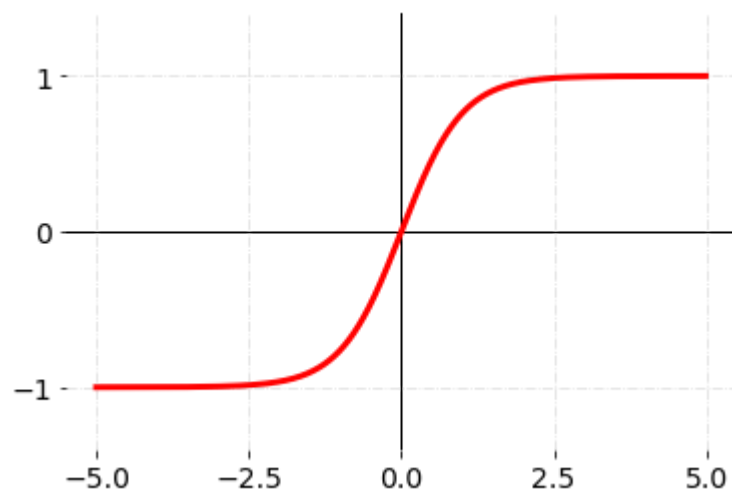
```
In [5]: plot(sigmoid, yaxis=(-0.4, 1.4))
```



## Tan h

```
In [6]: def tanh(x):  
        return 2 / (1 + np.exp(-2 * x)) - 1
```

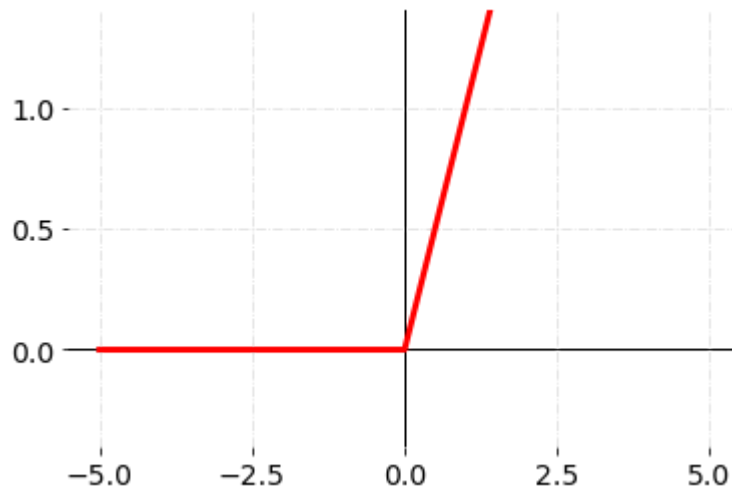
```
In [7]: plot(tanh)
```



## Rectified Linear Units (ReLU)

```
In [8]: relu = np.vectorize(lambda x: x if x > 0 else 0, otypes=[np.float])
```

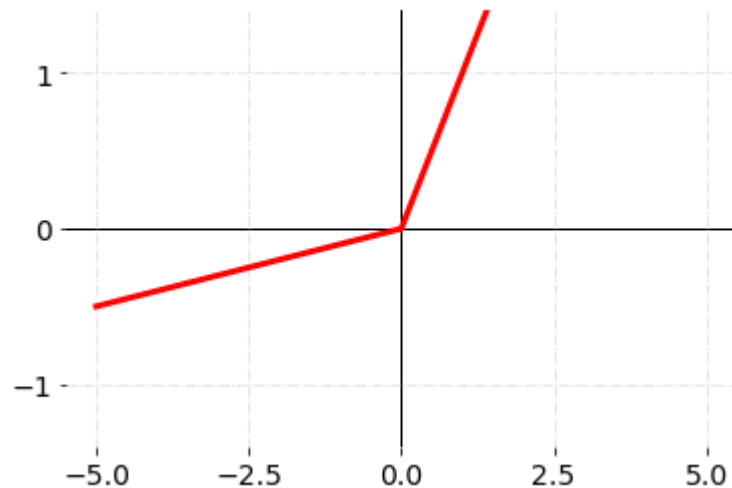
```
In [9]: plot(relu, yaxis=(-0.4, 1.4))
```



## Leaky Rectified Linear Units (Leaky ReLU)

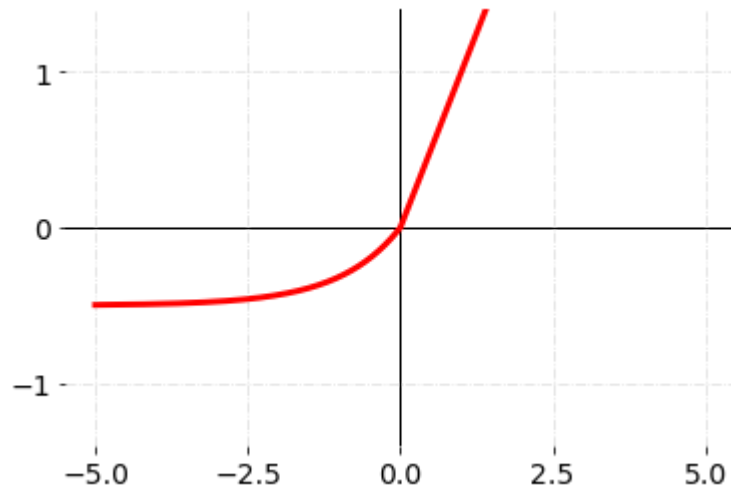
```
In [10]: leaky_relu = np.vectorize(lambda x: max(0.1 * x, x), otypes=[np.float])
```

```
In [11]: plot(leaky_relu)
```



## Exponential Linear Units (ELU)

```
In [12]: elu = np.vectorize(lambda x: x if x > 0 else 0.5 * (np.exp(x) - 1), otypes=[np.float64])  
plot(elu)
```



**Name :- Prekshita patil**

**Registration Number :- 20MAI0073**

```
In [1]: import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
In [2]: def processImage(image):
        image = cv2.imread(image)
        image = cv2.cvtColor(src=image, code=cv2.COLOR_BGR2GRAY)
        return image
```

```

In [3]: 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 + yKernShape]).sum()
            except:
                break

return output

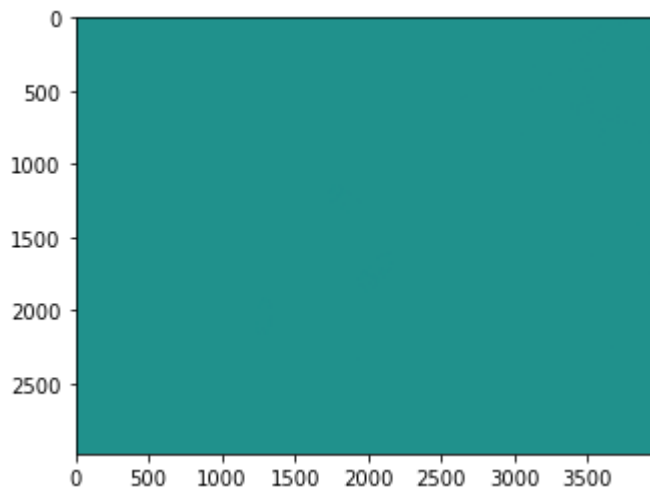
```

```
In [4]: if __name__ == '__main__':
# Grayscale Image
image = processImage('guillaume-baudusseau-2ZMjnR2d3qk-unsplash.jpg')

# Edge Detection Kernel
kernel = np.array([[ -1,  -1,  -1], [ -1,  8,  -1], [ -1,  -1,  -1]])

output = convolve2D(image, kernel, padding=2)
plt.imshow(output)
plt.show()
```

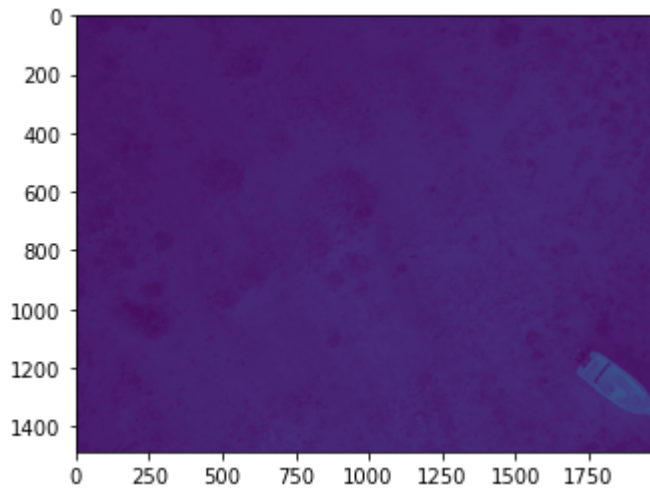
```
[[ 0.  0.  0. ...  0.  0.  0.]
 [ 0.  0.  0. ...  0.  0.  0.]
 [ 0.  0. 69. ... 101.  0.  0.]
 ...
 [ 0.  0. 63. ... 106.  0.  0.]
 [ 0.  0.  0. ...  0.  0.  0.]
 [ 0.  0.  0. ...  0.  0.  0.]]
```



## Taking stride as 2 and kernel size as 5\*5

```
In [5]: stride=2
filter_size=5
```

```
In [6]: kernel = np.array([[1, 1, 1,1,1], [1, 1, 1,1,1], [1, 1, 1,1,1],[1, 1, 1,1,1], [1,
# Convolve and Save Output
output2 = convolve2D(image, kernel,strides=2)
plt.imshow(output2)
plt.show()
```



```
In [7]: output2
```

```
Out[7]: array([[1637.,    0., 1643., ...,    0., 2235.,    0.],
 [    0.,    0.,    0., ...,    0.,    0.,    0.],
 [1617.,    0., 1637., ...,    0., 2236.,    0.],
 ...,
 [    0.,    0.,    0., ...,    0.,    0.,    0.],
 [1662.,    0., 1550., ...,    0., 1613.,    0.],
 [    0.,    0.,    0., ...,    0.,    0.,    0.]])
```

## Taking stride as 1 but making an image padded with 0 padding

```
In [8]: import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers.convolutional import Conv2D
import cv2
from matplotlib import pyplot as plt
```

```
In [9]: def processImage(image):

    image = cv2.imread(image)
    image = cv2.cvtColor(src=image, code=cv2.COLOR_BGR2GRAY)
    return image
```



```

In [10]: 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

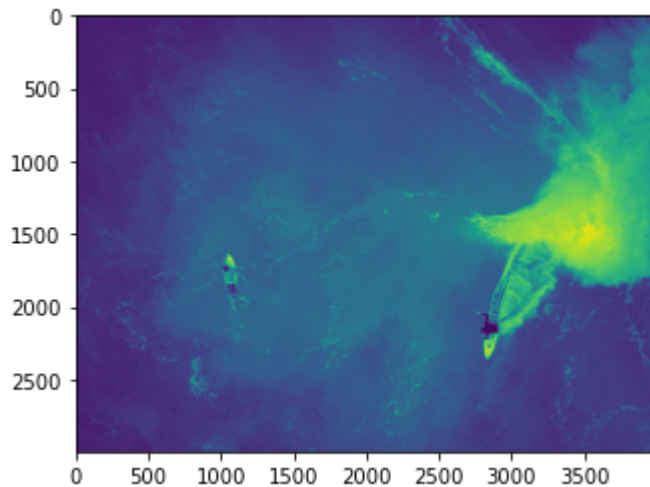
# Iterate through image
for y in range(image.shape[1]):
    # Exit Convolution
    if y > image.shape[1] - yKernShape:
        break
    # Only Convolve if y has gone down by the specified Strides
    if y % strides == 0:
        for x in range(image.shape[0]):
            # Go to next row once kernel 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 + yKernShape]).sum()
            except:
                break

return output

```

```
In [11]: image = processImage('michael-olsen-xbRe2QwIi-8-unsplash.jpg')
kernel = np.array([[1, 1, 1], [1, 1, 1], [1, 1, 1]])
output3 = convolve2D(image, kernel, padding=0)
plt.imshow(output3)
```

Out[11]: <matplotlib.image.AxesImage at 0x12a219619a0>



```
In [12]: output3
```

```
Out[12]: array([[180., 170., 161., ..., 495., 487., 481.],
                [182., 174., 165., ..., 494., 488., 482.],
                [173., 169., 162., ..., 491., 488., 484.],
                ...,
                [217., 215., 208., ..., 386., 391., 391.],
                [213., 212., 204., ..., 390., 395., 390.],
                [213., 212., 204., ..., 389., 396., 388.]])
```

## Calculating metrics i.e. entropy between original image and filtered image

```
In [13]: import skimage.measure
entropy_original_image = skimage.measure.shannon_entropy(image)
```

```
In [14]: print(entropy_original_image)
```

```
6.9196791409088245
```

```
In [15]: import skimage.measure  
entropy_for_1_stride = skimage.measure.shannon_entropy(output)
```

```
In [16]: print(entropy_for_1_stride)
```

```
6.9588006035582275
```

```
In [17]: import skimage.measure  
entropy_for_2_stride = skimage.measure.shannon_entropy(output2)
```

```
In [18]: print(entropy_for_2_stride)
```

```
3.491078469771006
```

```
In [19]: import skimage.measure  
entropy_for_0_padding = skimage.measure.shannon_entropy(output3)
```

```
In [20]: print(entropy_for_0_padding)
```

```
10.074482698094565
```