Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Artificial Intelligence & Machine Learning
Academic Year 2023-2024

Name: Abhay Mathur **Sapid:** 60017210016
Batch: A1

Experiment No. 1

Aim: Image Assessment with NumPy and OpenCV

Objective: Develop a program to perform Basic Image Processing Operations in Python

Theory:

Computer vision is a process by which we can understand the images and videos how they are stored and how we can manipulate and retrieve data from them. Computer Vision is the base or mostly used for Artificial Intelligence. Computer-Vision is playing a major role in self-driving cars, robotics as well as in photo correction apps.

OpenCV is the huge open-source library for the computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today's systems. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. When it integrated with various libraries, such as NumPy, python can process the OpenCV array structure for analysis. To Identify image pattern and its various features we use vector space and perform mathematical operations on these features.

Python Imaging Library (expansion of PIL) is the de facto image processing package for Python language. It incorporates lightweight image processing tools that aids in editing, creating, and saving images

NumPy also called Numerical Python is an amazing library open-source Python library for data manipulation and scientific computing. It is used in the domain of linear algebra, Fourier transforms, matrices, and the data science field. which is used.

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack.

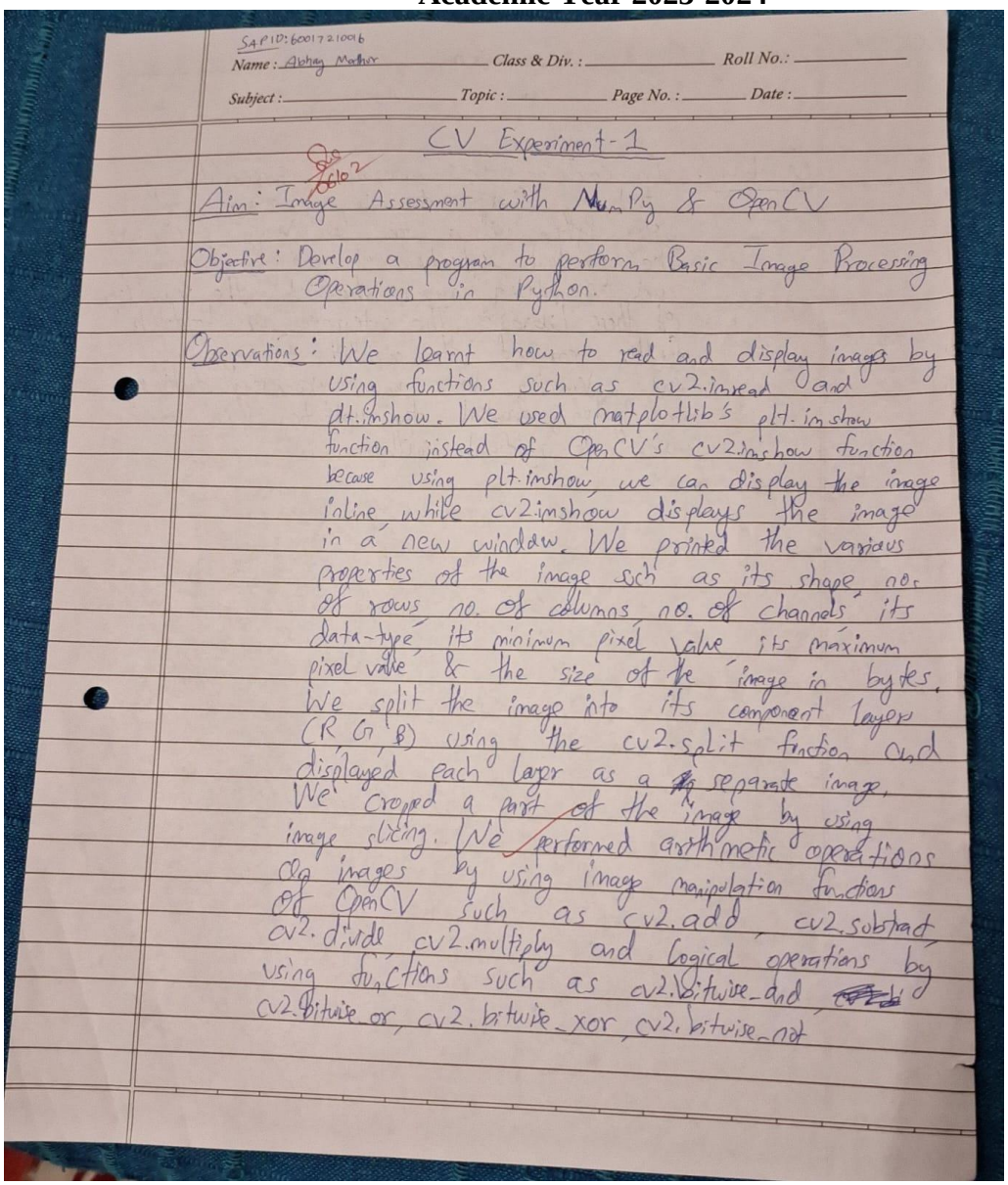
The image module in matplotlib library is used for working with images in Python. The image module also includes two useful methods which are imread which is used to read images and imshow which is used to display the image

Problem Definition:

- Read an Image
- Display an Image
- Observe its properties
- Splitting the layers
- Convert in Grey Scale
- Crop an Image
- Arithmetic Operations
- Logical Operations

Observations:

Department of Artificial Intelligence & Machine Learning
Academic Year 2023-2024



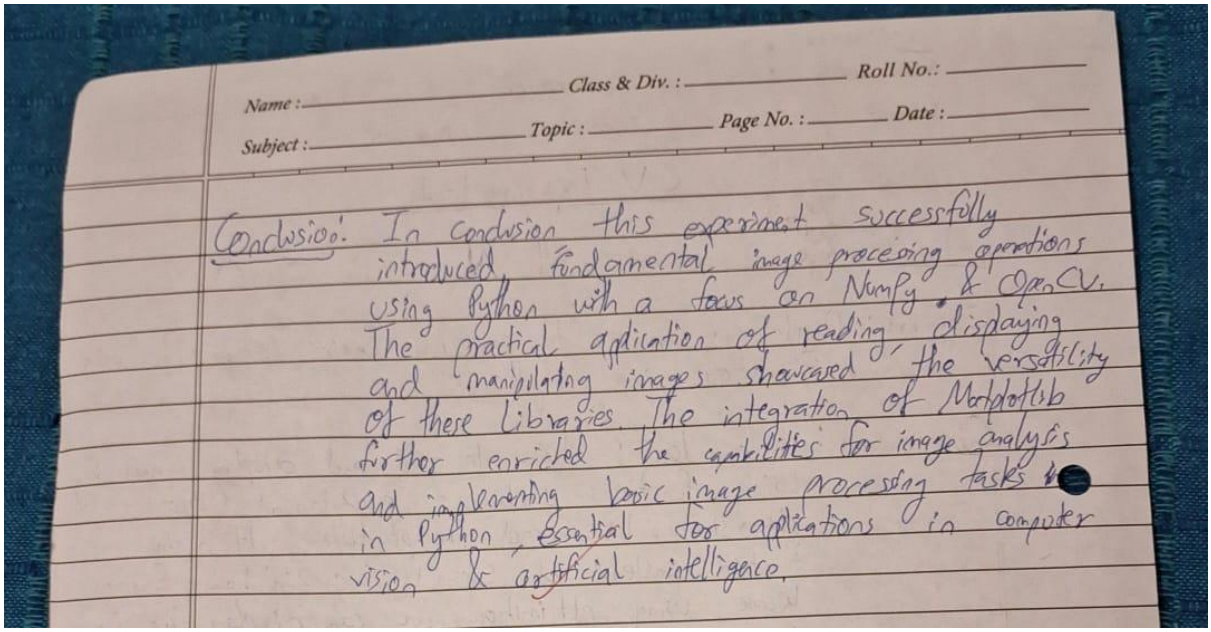
SAP ID: 60017210016
Name : Abhay Mathur Class & Div. : Roll No.:
Subject : Topic : Page No. : Date :

CV Experiment-1

Aim: Image Assessment with NumPy & OpenCV

Objective: Develop a program to perform Basic Image Processing Operations in Python.

Observations: We learnt how to read and display images by using functions such as cv2.imread and plt.imshow. We used matplotlib's plt.imshow function instead of OpenCV's cv2.imshow function because using plt.imshow, we can display the image inline while cv2.imshow displays the image in a new window. We printed the various properties of the image such as its shape, no. of rows, no. of columns, no. of channels, its data-type, its minimum pixel value, its maximum pixel value & the size of the image in bytes. We split the image into its component layers (R G B) using the cv2.split function and displayed each layer as a separate image. We cropped a part of the image by using image slicing. We performed arithmetic operations on images by using image manipulation functions of OpenCV such as cv2.add, cv2.subtract, cv2.divide, cv2.multiply and logical operations by using functions such as cv2.bitwise_and, cv2.bitwise_or, cv2.bitwise_xor, cv2.bitwise_not



Conclusion:

In conclusion, this experiment successfully introduced fundamental image processing operations using Python with a focus on NumPy and OpenCV. The practical application of reading, displaying, and manipulating images showcased the versatility of these libraries. The integration of Matplotlib further enriched the capabilities for image analysis and visualization. Overall, the experiment provided a solid foundation for understanding and implementing basic image processing tasks in Python, essential for applications in computer vision and artificial intelligence.

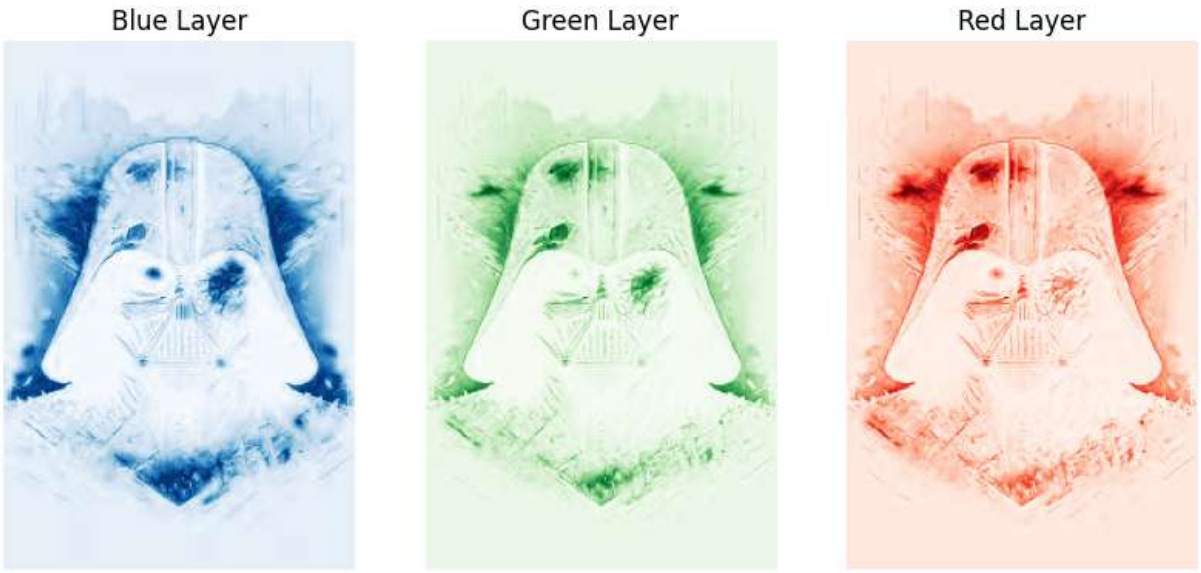
In [ ]:
```
import cv2
import matplotlib.pyplot as plt
```

In [ ]:
```
# Read the image
image = cv2.imread('test_image.jpg')

# Convert the image from BGR to RGB
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Display the image inline
plt.imshow(image)
plt.axis('off')
plt.show()
```

In [ ]:
```
# Print the properties of the image
print("Image shape:", image.shape)
print("Number of rows:", image.shape[0])
print("Number of columns:", image.shape[1])
print("Number of channels:", image.shape[2])
# Print the data type of the image
print("Data type:", image.dtype)

# Print the minimum and maximum pixel values
print("Minimum pixel value:", image.min())
print("Maximum pixel value:", image.max())

# Print the image size in bytes
print("Image size:", image.nbytes, "bytes")
```

```
Image shape: (275, 183, 3)
Number of rows: 275
Number of columns: 183
Number of channels: 3
Data type: uint8
Minimum pixel value: 0
Maximum pixel value: 255
Image size: 150975 bytes
```

In [ ]:
```python
# Split the image into its component layers
b, g, r = cv2.split(image)

# Display the component layers
plt.figure(figsize=(10, 5))

plt.subplot(1, 3, 1)
plt.imshow(b, cmap='Blues')
plt.title('Blue Layer')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(g, cmap='Greens')
plt.title('Green Layer')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(r, cmap='Reds')
plt.title('Red Layer')
plt.axis('off')

plt.show()
```



In [ ]:
```python
# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Display the grayscale image
plt.imshow(gray_image, cmap='gray')
```

```
plt.axis('off')
plt.show()
```

In [ ]:
```
# Define the coordinates of the region to crop
x = 100  # starting x-coordinate
y = 100  # starting y-coordinate
width = 200  # width of the cropped region
height = 200  # height of the cropped region

# Crop the image
cropped_image = image[y:y+height, x:x+width]

# Display the cropped image
plt.imshow(cropped_image)
plt.axis('off')
plt.show()
```

In [ ]:
```
import numpy as np

# Add a constant value to each pixel
addition_image = image + 50
subtraction_image = image - 50
division_image = image // 2
multiplication_image = image * 2

# Create a figure with subplots
fig, axes = plt.subplots(2, 2, figsize=(10, 10))

# Display the addition image
axes[0, 0].imshow(addition_image)
axes[0, 0].set_title('Addition Image')
axes[0, 0].axis('off')

# Display the subtraction image
axes[0, 1].imshow(subtraction_image)
axes[0, 1].set_title('Subtraction Image')
axes[0, 1].axis('off')

# Display the division image
axes[1, 0].imshow(division_image)
axes[1, 0].set_title('Division Image')
axes[1, 0].axis('off')

# Display the multiplication image
axes[1, 1].imshow(multiplication_image)
axes[1, 1].set_title('Multiplication Image')
axes[1, 1].axis('off')

# Adjust the spacing between subplots
```

```
plt.tight_layout()

# Show the plot
plt.show()
```

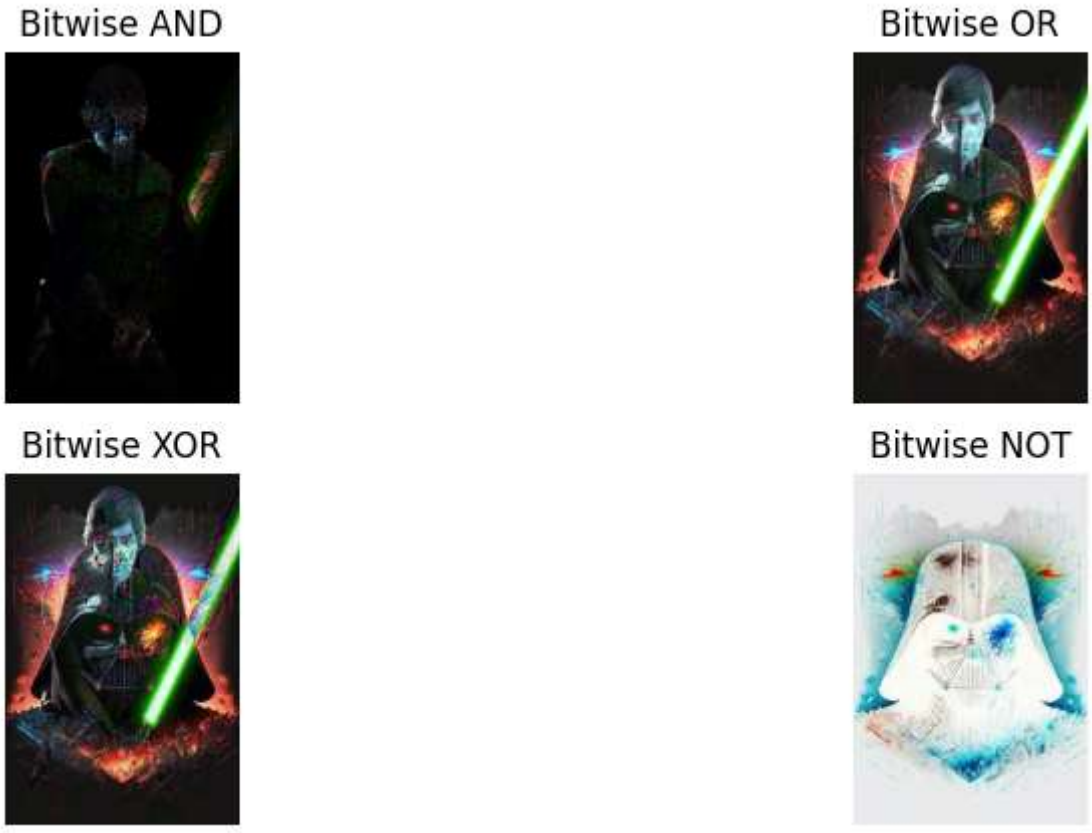
In [ ]:
```
img2 = cv2.imread('test_image_2.jpg')
img2 = cv2.resize(img2, (image.shape[1], image.shape[0]))
```

```
images_added = cv2.add(image, img2)
images_subtracted = cv2.subtract(image, img2)
images_divided = cv2.divide(image, img2, dtype=cv2.CV_32F)
images_multiplied = cv2.multiply(image, img2)

images_added_result = np.clip(images_added, 0, 255).astype(np.uint8)
images_subtracted_result = np.clip(images_subtracted, 0, 255).astype(np.uint8)
images_divided_result = np.clip(images_divided, 0, 255).astype(np.uint8)
images_multiplied_result = np.clip(images_multiplied, 0, 255).astype(np.uint8)


# Create a figure with subplots
fig, axes = plt.subplots(2, 2, figsize=(10, 10))

# Display the addition image
axes[0, 0].imshow(images_added_result)
axes[0, 0].set_title('Addition Image')
axes[0, 0].axis('off')

# Display the subtraction image
axes[0, 1].imshow(images_subtracted_result)
axes[0, 1].set_title('Subtraction Image')
axes[0, 1].axis('off')

# Display the division image
axes[1, 0].imshow(images_divided_result)
axes[1, 0].set_title('Division Image')
axes[1, 0].axis('off')

# Display the multiplication image
axes[1, 1].imshow(images_multiplied_result)
axes[1, 1].set_title('Multiplication Image')
axes[1, 1].axis('off')

# Adjust the spacing between subplots
plt.tight_layout()

# Show the plot
plt.show()
```

```
C:\Users\a21ma\AppData\Local\Temp\ipykernel_91712\536329118.py:11: RuntimeWarning: i
nvalid value encountered in cast
  images_divided_result = np.clip(images_divided, 0, 255).astype(np.uint8)
```

In [ ]:
```
# Perform bitwise AND operation
result_and = cv2.bitwise_and(image, img2)

# Perform bitwise OR operation
result_or = cv2.bitwise_or(image, img2)

# Perform bitwise XOR operation
result_xor = cv2.bitwise_xor(image, img2)
```

```
# Perform bitwise NOT operation
result_not = cv2.bitwise_not(image)

# Display the results
plt.figure(figsize=(10, 5))

plt.subplot(2, 2, 1)
plt.imshow(result_and)
plt.title('Bitwise AND')
plt.axis('off')

plt.subplot(2, 2, 2)
plt.imshow(result_or)
plt.title('Bitwise OR')
plt.axis('off')

plt.subplot(2, 2, 3)
plt.imshow(result_xor)
plt.title('Bitwise XOR')
plt.axis('off')

plt.subplot(2, 2, 4)
plt.imshow(result_not)
plt.title('Bitwise NOT')
plt.axis('off')

plt.show()
```



Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Artificial Intelligence & Machine Learning
Academic Year 2023-2024

Name: Abhay Mathur **Sapid:** 60017210016
Batch: A1

Experiment No. 2

Aim: Image Transformations

Objective: Develop a program to perform different Image Transformations

Theory:

Image Transformation involves the transformation of image data to retrieve information from the image or preprocess the image for further usage

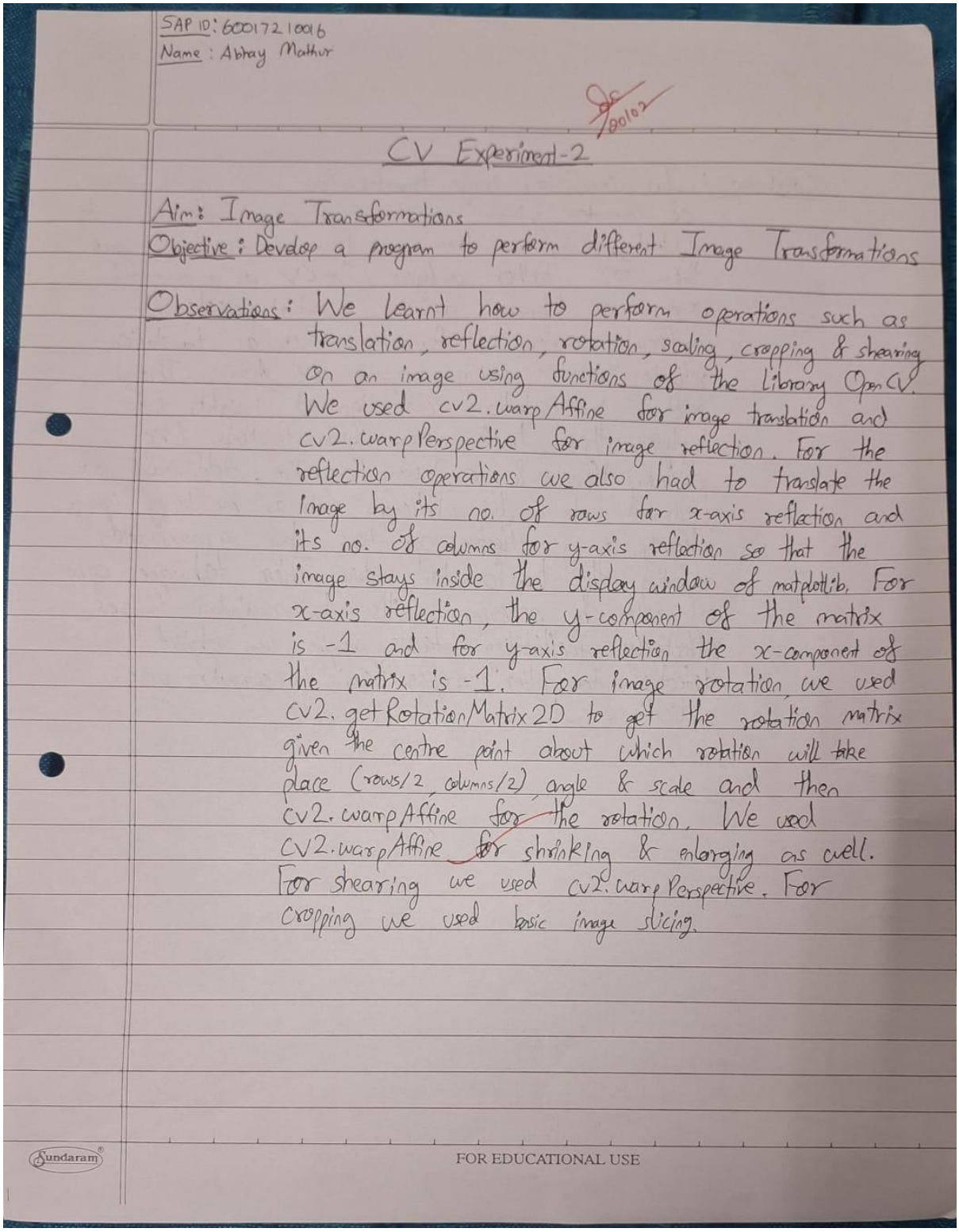
OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products. By using it, one can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as NumPy, Python is capable of processing the OpenCV array structure for analysis.

Problem Definition

- Image Translation
- Reflection
- Rotation
- Scaling
- Cropping
- Shearing in x-axis
- Shearing in y-axis

Department of Artificial Intelligence & Machine Learning
Academic Year 2023-2024

Observations:



SAP ID: 60017210016
Name : Abhay Mathur

9/01/02

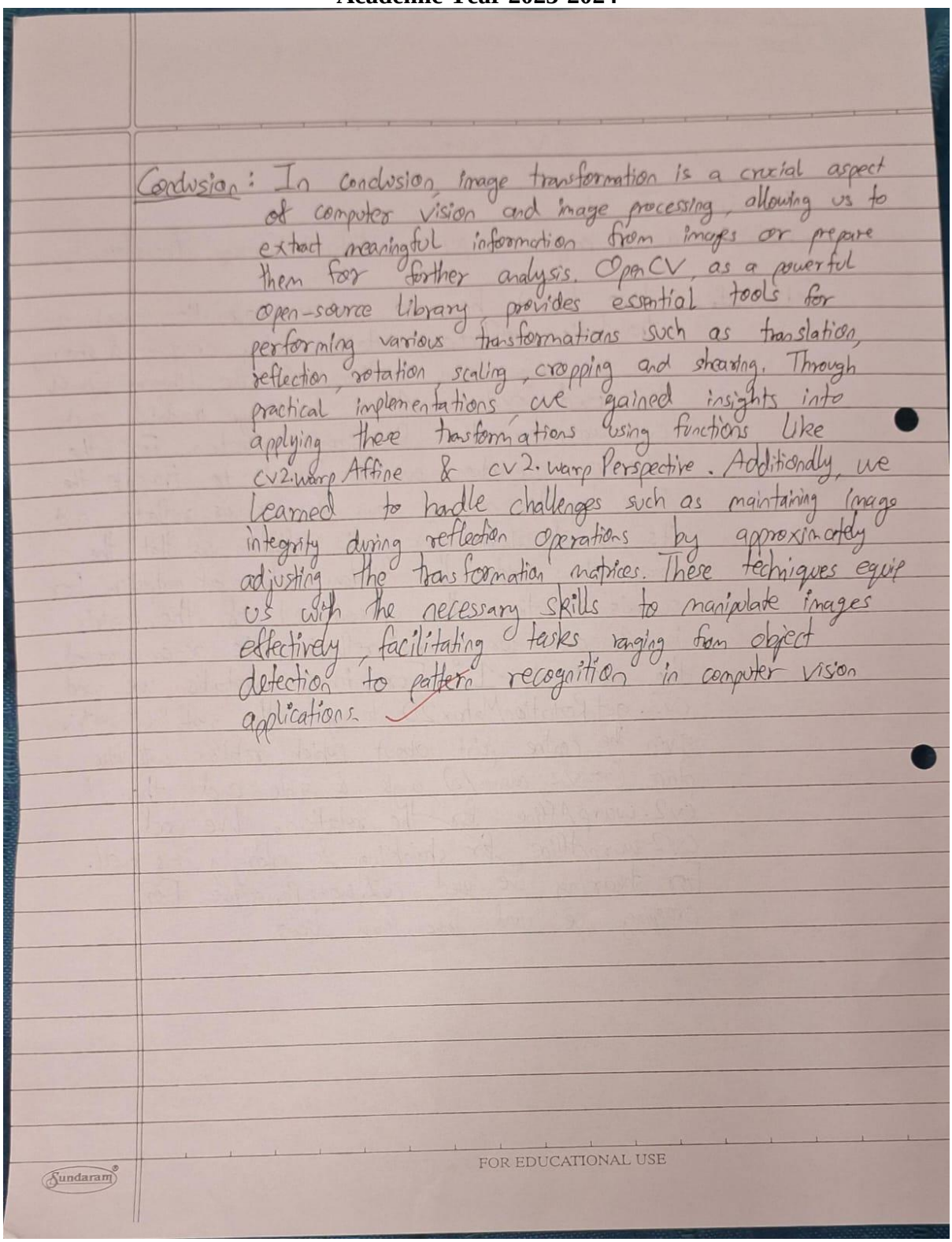
CV Experiment-2

Aim: Image Transformations
Objective: Develop a program to perform different Image Transformations

Observations: We learnt how to perform operations such as translation, reflection, rotation, scaling, cropping & shearing on an image using functions of the library OpenCV. We used cv2.warpAffine for image translation and cv2.warpPerspective for image reflection. For the reflection operations we also had to translate the image by its no. of rows for x-axis reflection and its no. of columns for y-axis reflection so that the image stays inside the display window of matplotlib. For x-axis reflection, the y-component of the matrix is -1 and for y-axis reflection the x-component of the matrix is -1. For image rotation, we used cv2.getRotationMatrix2D to get the rotation matrix given the centre point about which rotation will take place (rows/2, columns/2), angle & scale and then cv2.warpAffine for the rotation. We used cv2.warpAffine for shrinking & enlarging as well. For shearing we used cv2.warpPerspective. For cropping we used basic image slicing.

Sundaram

FOR EDUCATIONAL USE



Conclusion: In conclusion, image transformation is a crucial aspect of computer vision and image processing, allowing us to extract meaningful information from images or prepare them for further analysis. OpenCV, as a powerful open-source library, provides essential tools for performing various transformations such as translation, reflection, rotation, scaling, cropping and shearing. Through practical implementations, we gained insights into applying these transformations using functions like cv2.warpAffine & cv2.warpPerspective. Additionally, we learned to handle challenges such as maintaining image integrity during reflection operations by appropriately adjusting the transformation matrices. These techniques equip us with the necessary skills to manipulate images effectively, facilitating tasks ranging from object detection to pattern recognition in computer vision applications.

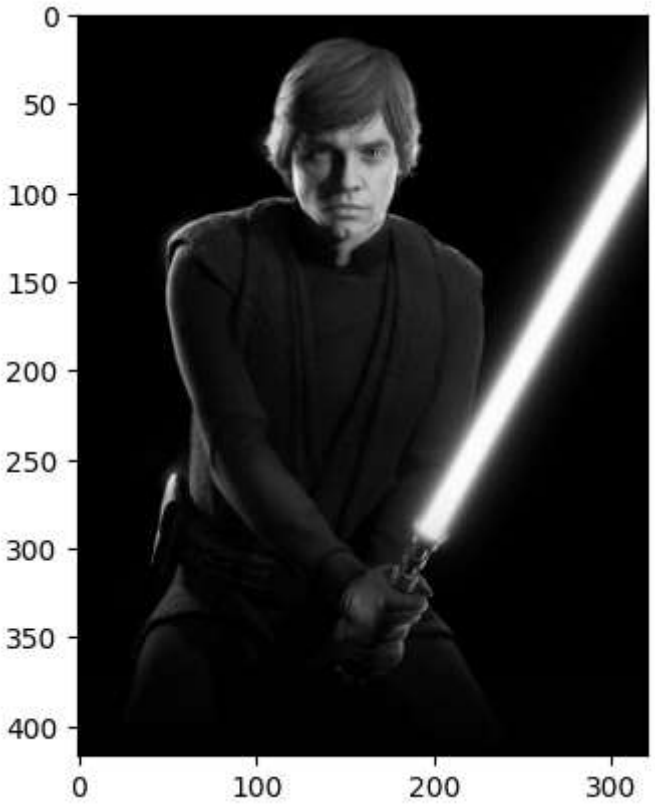
Conclusion:

In conclusion, image transformation is a crucial aspect of computer vision and image processing, allowing us to extract meaningful information from images or prepare them for further analysis. OpenCV, as a powerful open-source library, provides essential tools for performing various transformations such as translation, reflection, rotation, scaling, cropping and shearing. Through practical implementations, we gained insights into applying these transformations using functions like cv2.warpAffine and cv2.warpPerspective. Additionally, we learned to handle challenges such as maintaining image integrity during reflection operations by appropriately adjusting the transformation matrices. These techniques equip us with the necessary skills to manipulate images effectively, facilitating tasks ranging from object detection to pattern recognition in computer vision applications.

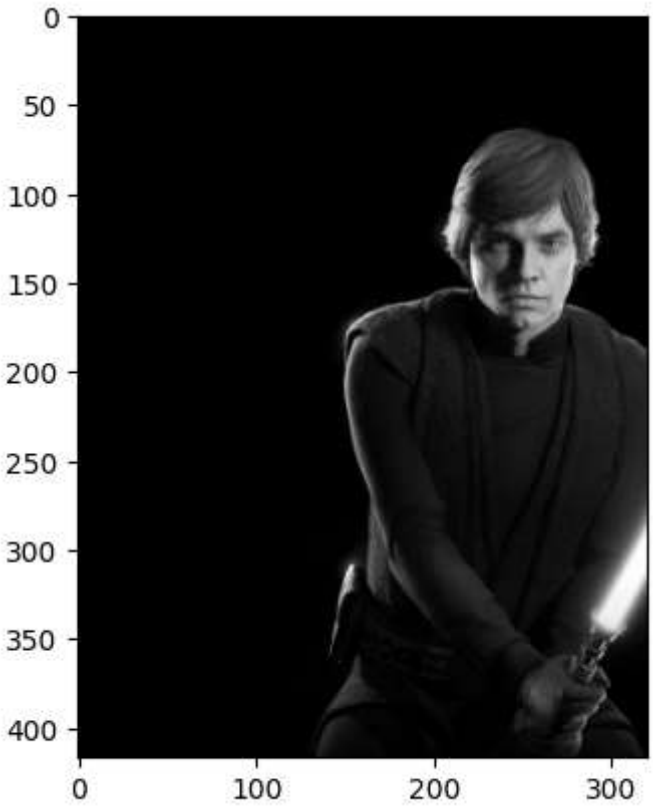
Department of Artificial Intelligence & Machine Learning
Academic Year 2023-2024

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

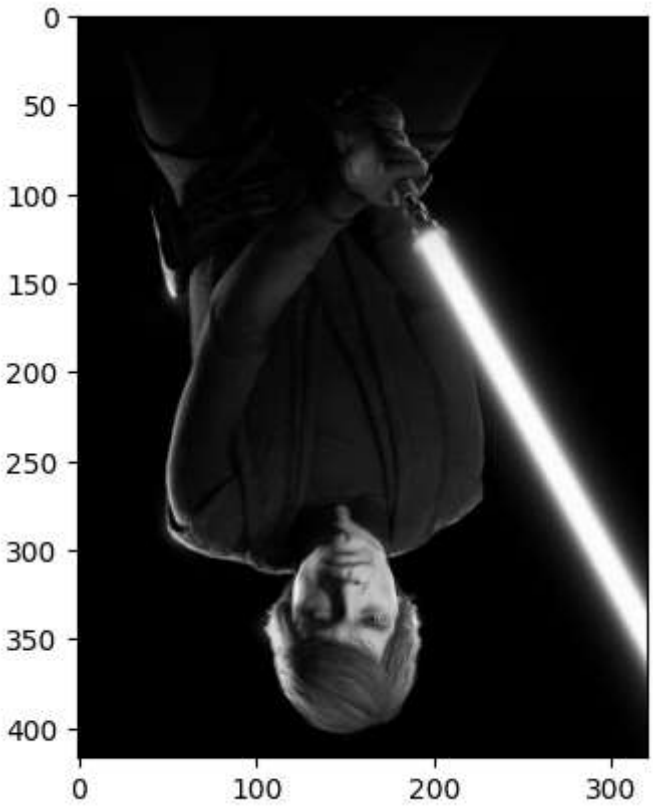
In [ ]:
```
img = cv2.imread('photo.jpg',0) # the '0' means read as grayscale
rows, cols = img.shape
plt.imshow(img, cmap='gray')
plt.show()
```



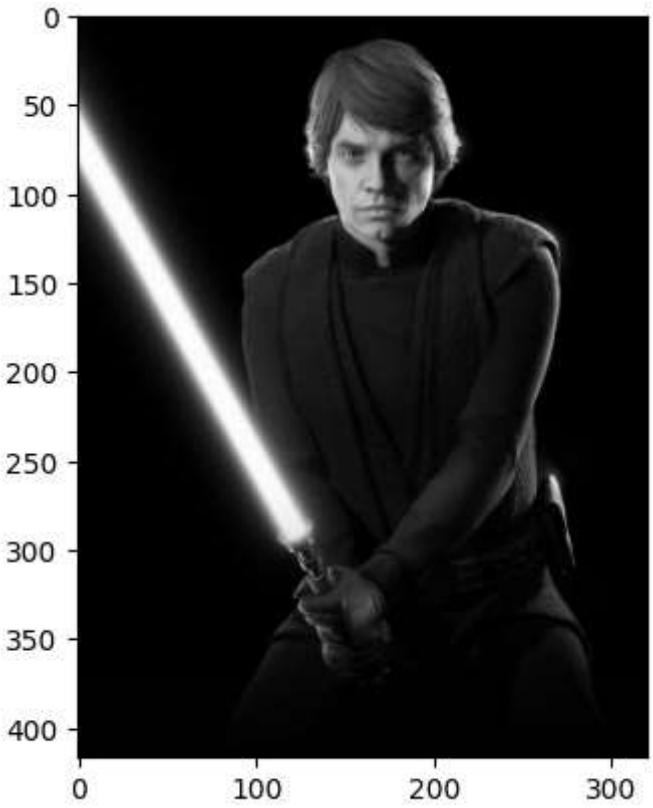
In [ ]:
```
M = np.float32([[1, 0, 100],
                [0, 1, 50]]) # where tx = 100 and ty = 50
translated_image = cv2.warpAffine(img, M, (cols, rows))
plt.imshow(translated_image, cmap='gray')
plt.show()
```



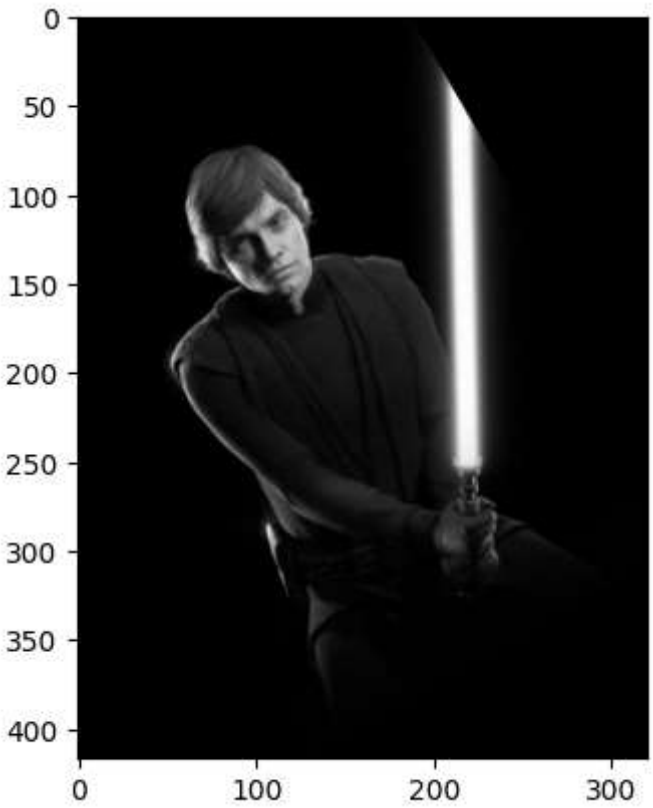
In [ ]:
```
M = np.float32([[1,  0, 0],
                [0, -1, rows], #translated down by 'rows' so we can see it in the d
                [0,  0, 1]])
X_reflected_img = cv2.warpPerspective(img, M,(int(cols),int(rows)))
plt.imshow(X_reflected_img, cmap='gray')
plt.show()
```



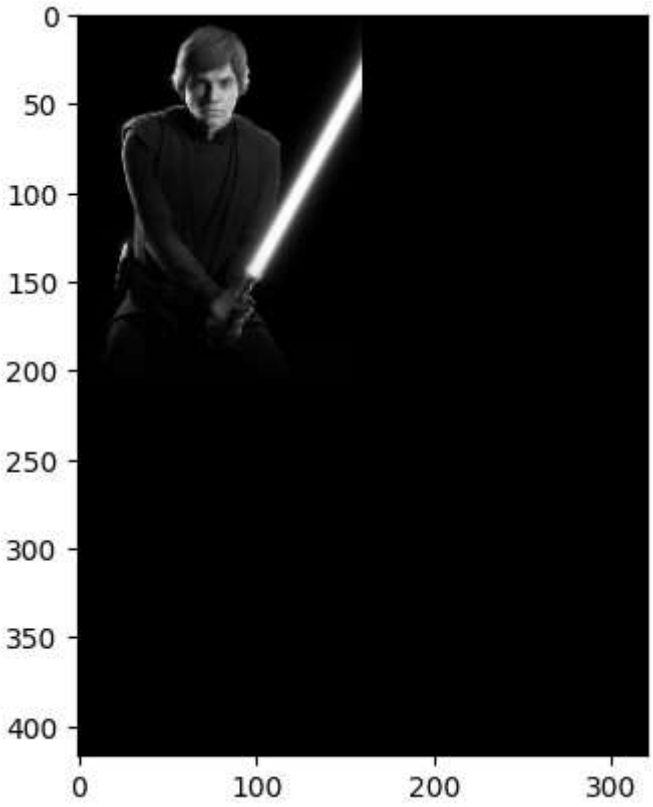
In [ ]:
```
M = np.float32([[-1, 0, cols], #translated right by 'cols' so we can see it in the
                [0, 1, 0],
                [0, 0, 1]])
Y_reflected_img = cv2.warpPerspective(img, M,(int(cols),int(rows)))
plt.imshow(Y_reflected_img, cmap='gray')
plt.show()
```



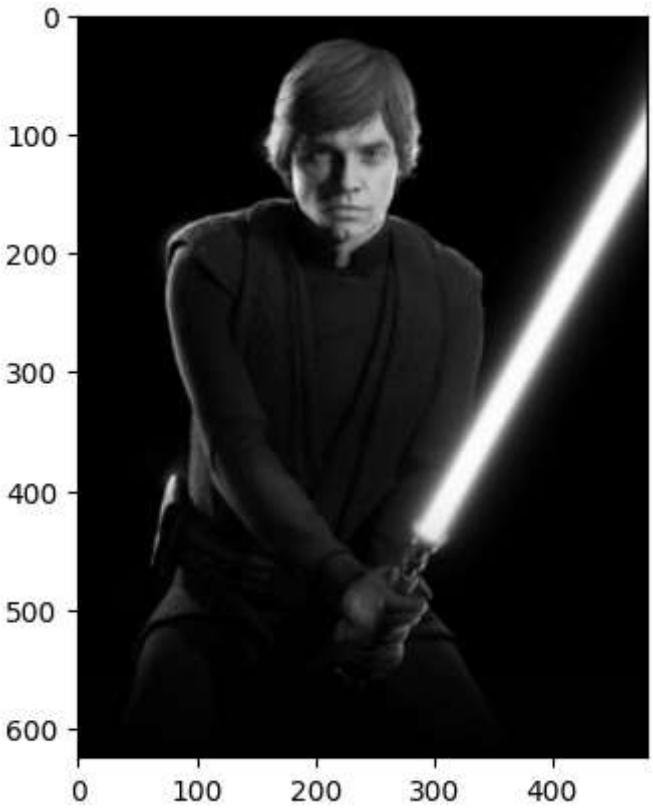
In [ ]:
```
img_rotation = cv2.warpAffine(img,cv2.getRotationMatrix2D((cols/2, rows/2),30,0.8),
plt.imshow(img_rotation, cmap='gray')
plt.show()
```



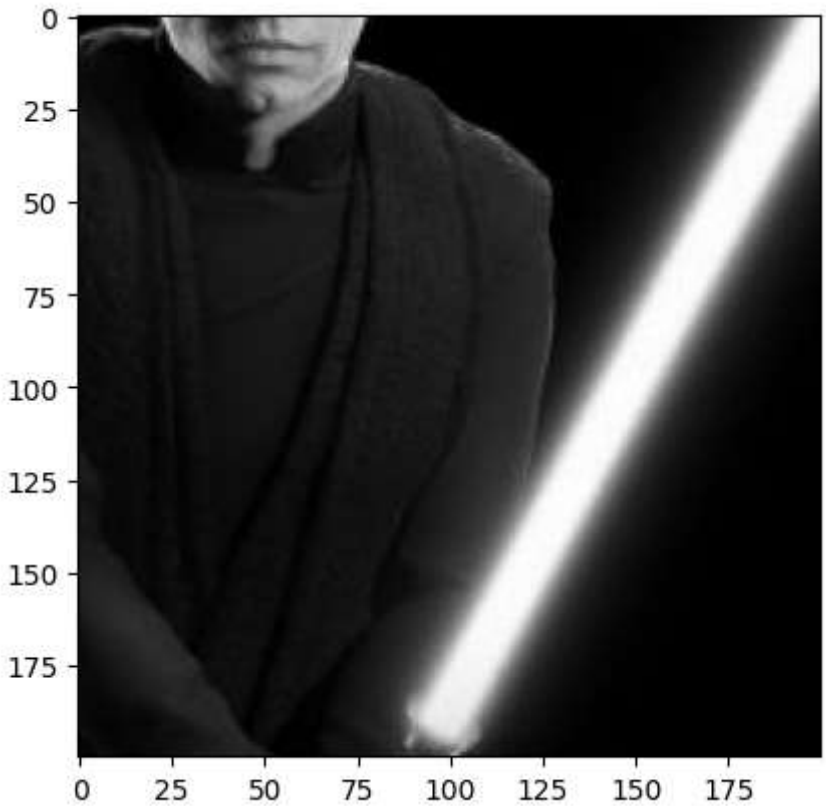
In [ ]:
```
M=np.float32([[0.5,0,0],
              [0,0.5,0]])
img_shrinked = cv2.warpAffine(img,M,(int(cols), int(rows)))
plt.imshow(img_shrinked,cmap='gray')
plt.show()
```



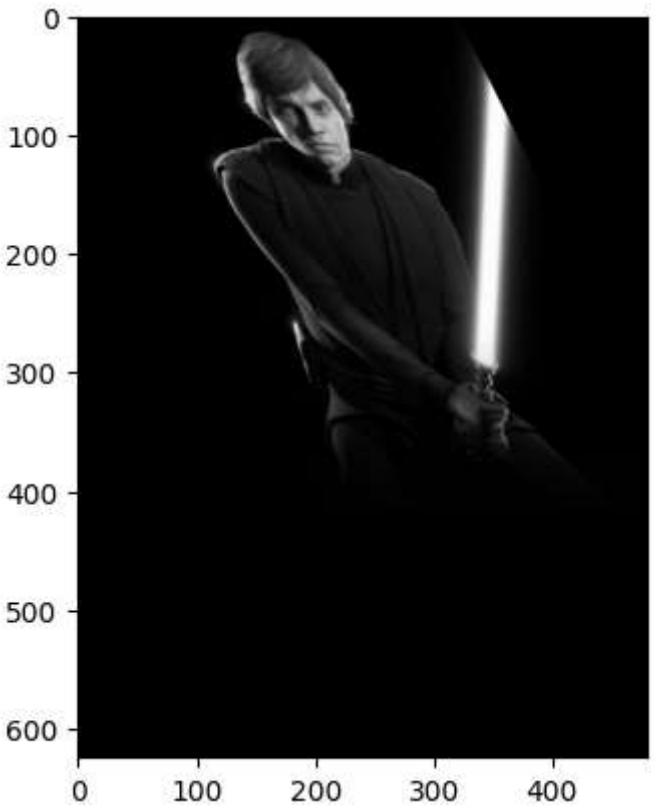
In [ ]:
```
M=np.float32([[1.5,0,0],[0,1.5,0]])
img_enlarged = cv2.warpAffine(img,M,(int(cols*1.5), int(rows*1.5)))
plt.imshow(img_enlarged,cmap='gray')
plt.show()
```



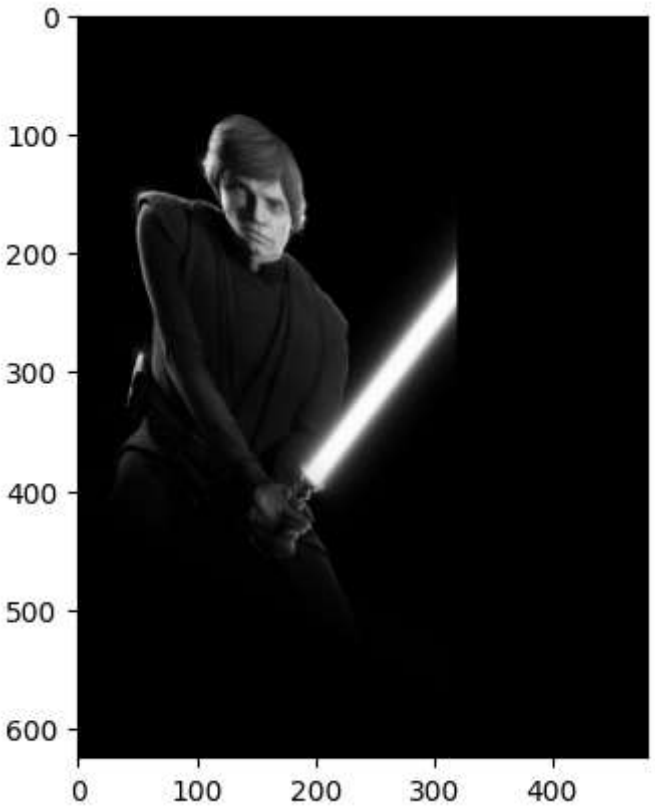
In [ ]:
```
cropped_img = img[100:300, 100:300]
plt.imshow(cropped_img,cmap='gray')
plt.show()
```



In [ ]:
```
M = np.float32([[1, 0.5, 0], [0, 1, 0], [0, 0, 1]])
x_sheared_img = cv2.warpPerspective(img, M, (int(cols*1.5), int(rows*1.5)))
plt.imshow(x_sheared_img,cmap='gray')
plt.show()
```

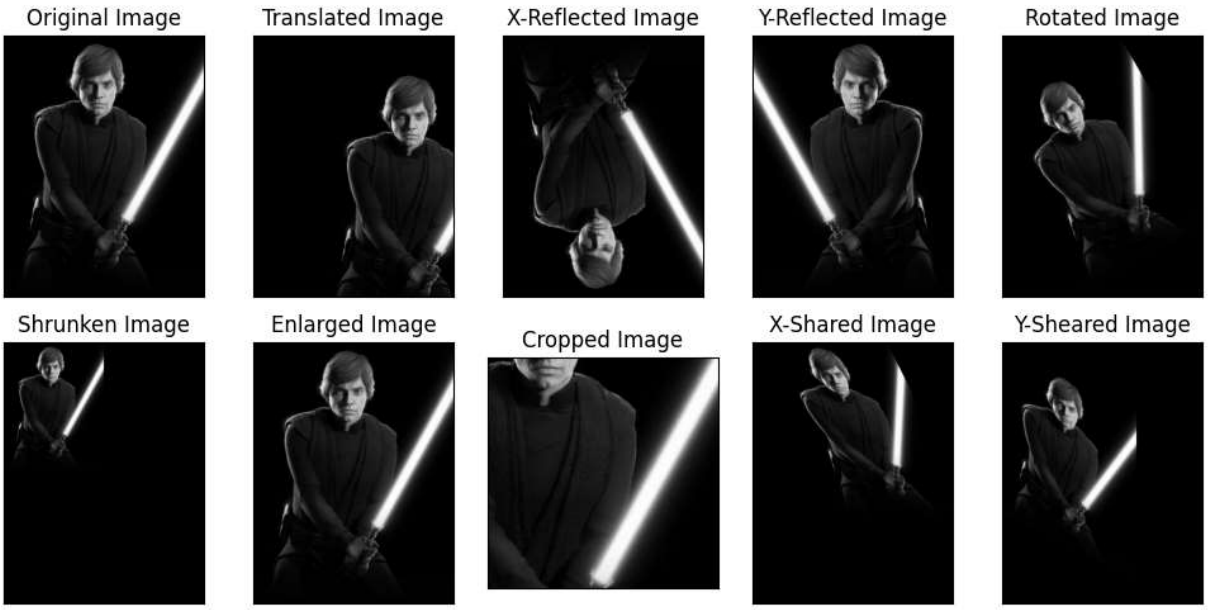


In [ ]:
```
M = np.float32([[1,   0, 0], [0.5, 1, 0], [0,   0, 1]])
y_sheared_img = cv2.warpPerspective(img, M, (int(cols*1.5), int(rows*1.5)))
plt.imshow(y_sheared_img,cmap='gray')
plt.show()
```



In [ ]:
```
fig, ax = plt.subplots(2,5, figsize=(10, 5))
ax[0][0].imshow(img, cmap='gray')
ax[0][1].imshow(translated_image, cmap='gray')
ax[0][2].imshow(X_reflected_img, cmap='gray')
ax[0][3].imshow(Y_reflected_img, cmap='gray')
ax[0][4].imshow(img_rotation, cmap='gray')
ax[1][0].imshow(img_shrinked, cmap='gray')
ax[1][1].imshow(img_enlarged, cmap='gray')
ax[1][2].imshow(cropped_img, cmap='gray')
ax[1][3].imshow(x_sheared_img, cmap='gray')
ax[1][4].imshow(y_sheared_img, cmap='gray')

ax[0][0].set_title('Original Image')
ax[0][1].set_title('Translated Image')
ax[0][2].set_title('X-Reflected Image')
ax[0][3].set_title('Y-Reflected Image')
ax[0][4].set_title('Rotated Image')
ax[1][0].set_title('Shrunken Image')
ax[1][1].set_title('Enlarged Image')
ax[1][2].set_title('Cropped Image')
ax[1][3].set_title('X-Shared Image')
ax[1][4].set_title('Y-Sheared Image')

for ax in ax.flat:
    # Remove ticks on both axes
    ax.set_xticks([])
    ax.set_yticks([])

# plt.axis('off')
```

```
plt.tight_layout() # to make the images fit better in the display window, preventin
plt.show()
```



Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Artificial Intelligence & Machine Learning
Academic Year 2023-2024

Name: Abhay Mathur **Sapid:** 60017210016
Batch: A1

Experiment No. 3

Aim: Image Denoising

Objective: Develop a program to add noise & remove it by different operators

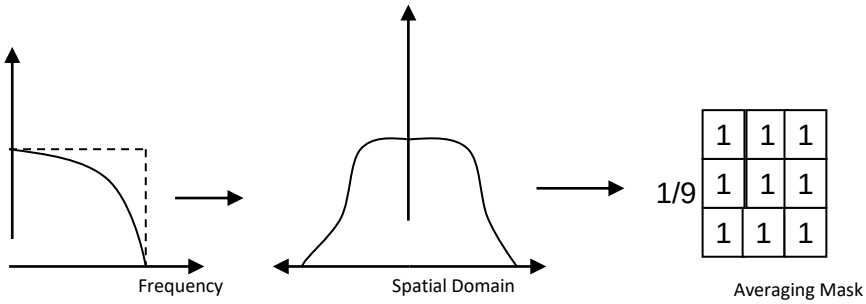
Theory:

Smoothing

Low pass filtering removes the high frequency content from the image. It is used to remove the noise present in the image which is generally a high frequency signal.

1. Averaging Filter

If an image has Gaussian noise, a low pass averaging filter is used to remove the noise. The frequency response and spatial response is show below.



From spatial response we generate the mask that would give us the low pass filtering operation. Here all the coefficients are positive and the multiplying factor for a M X N matrix is 1 / (M X N).

2. Median Filtering

The averaging filter removes the noise by filtering it till it is no longer seen. But in the process it also blurs the edges. If e use averaging filter to remove salt and pepper noise from an image it will blur the noise but will also ruin the edges. Hence when we need to remove salt and pepper noise we use non-linear filter known as Median Filter. They are also called order statistical filters because their response is based on the ordering and ranking of the pixels contained within the mask.

Image Denoising in OpenCV

OpenCV provides four variations of this technique.

cv.fastNlMeansDenoising() - works with a single grayscale images

cv.fastNlMeansDenoisingColored() - works with a color image.

cv.fastNlMeansDenoisingMulti() - works with image sequence captured in short period of time (grayscale images)

cv.fastNlMeansDenoisingColoredMulti() - same as above, but for color images.

Common arguments are:

h : parameter deciding filter strength. Higher h value removes noise better, but removes details of image also. (10 is ok)

hForColorComponents : same as h, but for color images only. (normally same as h)

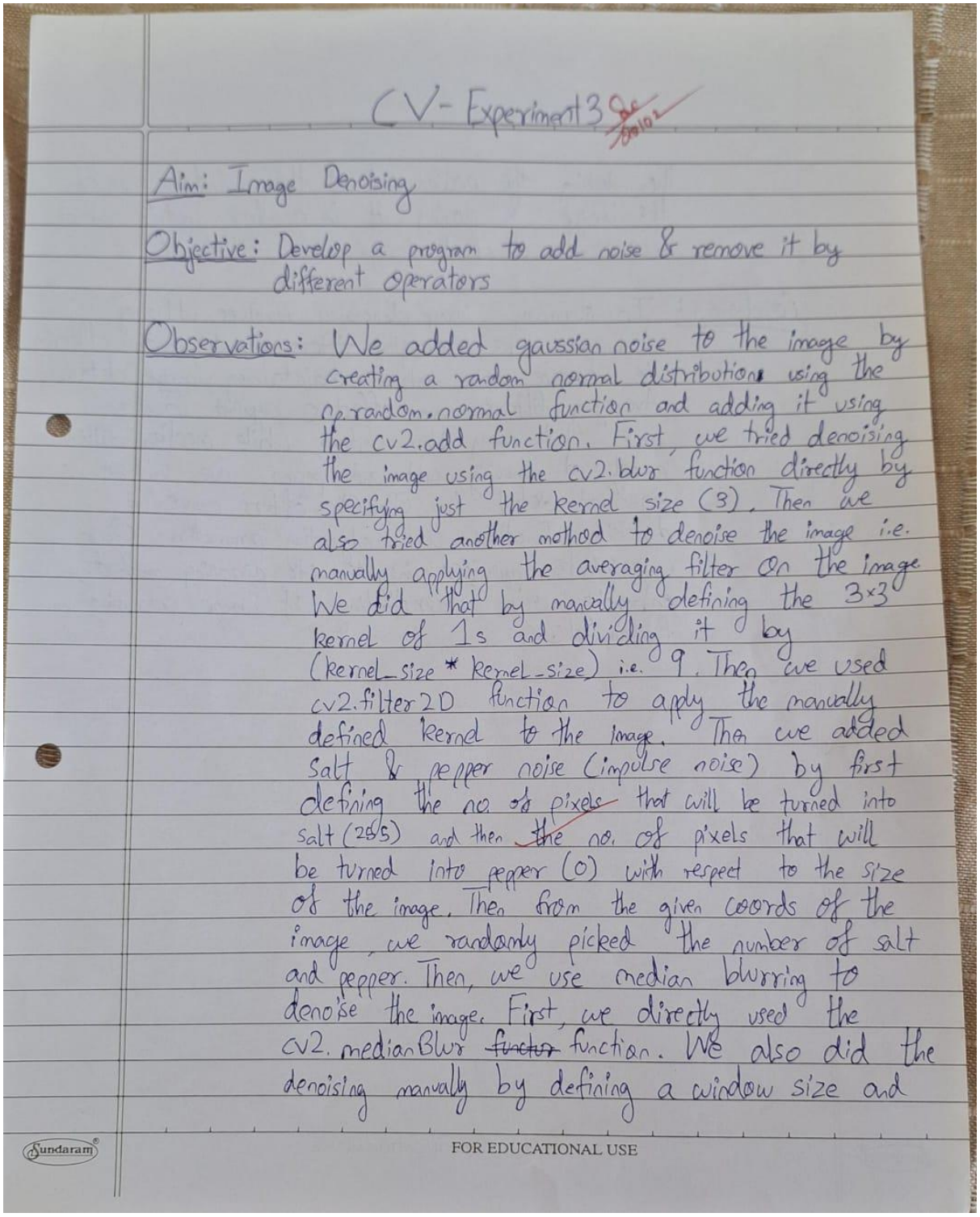
templateWindowSize : should be odd. (recommended 7)

searchWindowSize : should be odd. (recommended 21)

Problem Definition

- Add Noise & Remove it using different Operators
- Use OpenCV to denoise Images

Observations



CV- Experiment 3

Aim: Image Denoising

Objective: Develop a program to add noise & remove it by different operators

Observations: We added gaussian noise to the image by creating a random normal distribution using the np.random.normal function and adding it using the cv2.add function. First, we tried denoising the image using the cv2.blur function directly by specifying just the kernel size (3). Then we also tried another method to denoise the image i.e. manually applying the averaging filter on the image. We did that by manually defining the 3×3 kernel of 1s and dividing it by (kernel_size * kernel_size) i.e. 9. Then we used cv2.filter2D function to apply the manually defined kernel to the image. Then we added salt & pepper noise (impulse noise) by first defining the no. of pixels that will be turned into salt (255) and then the no. of pixels that will be turned into pepper (0) with respect to the size of the image. Then from the given coords of the image, we randomly picked the number of salt and pepper. Then, we use median blurring to denoise the image. First, we directly used the cv2.medianBlur ~~functor~~ function. We also did the denoising manually by defining a window size and

Sundaram

then taking the median out of the window size of the image & storing it in another matrix which becomes the denoised image once filled up.

Conclusion: In summary, image denoising involves utilizing techniques like averaging ~~filters~~ and median filtering to remove noise while maintaining image details. Averaging filters are effective against Gaussian noise but may blur edges, while median filters excel at removing salt and pepper noise without significant detail loss. OpenCV offers convenient denoising functions with adjusting parameters. Our experiments provided insights into denoising methods, enhancing our understading of image processing fundamentals.

Conclusion

In summary, image denoising involves utilizing techniques like averaging and median filtering to remove noise while maintaining image details. Averaging filters are effective against Gaussian noise but may blur edges, while median filters excel at removing salt and pepper noise without significant detail loss. OpenCV offers convenient denoising functions with adjustable parameters. Our experiments provided insights into denoising methods, enhancing our understanding of image processing fundamentals.

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

In [ ]:
```
image = cv2.imread('test_image3.jpg')
```

In [ ]:
```
gaussian_noise = np.random.normal(0, 0.6, image.shape).astype(np.uint8) # where 0 i
gaussian_noisy_image = cv2.add(image, gaussian_noise)

kernel_size = 3  # Define the kernel size
gaussian_denoised_image = cv2.blur(gaussian_noisy_image, (kernel_size, kernel_size)

kernel = np.ones((kernel_size, kernel_size), dtype=np.float32) / (kernel_size * ker
gaussian_denoised_image2 = cv2.filter2D(gaussian_noisy_image, -1, kernel)  # Convol

plt.figure(figsize=(5, 5))

plt.subplot(2, 2, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')

plt.subplot(2, 2, 2)
plt.imshow(cv2.cvtColor(gaussian_noisy_image, cv2.COLOR_BGR2RGB))
plt.title('Noisy Image')
plt.axis('off')

plt.subplot(2, 2, 3)
plt.imshow(cv2.cvtColor(gaussian_denoised_image, cv2.COLOR_BGR2RGB))
plt.title('Denoise Image 1')
plt.axis('off')

plt.subplot(2, 2, 4)
plt.imshow(cv2.cvtColor(gaussian_denoised_image2, cv2.COLOR_BGR2RGB))
plt.title('Denoise Image 2')
plt.axis('off')

plt.show()
```

In [ ]:
```
image2 = cv2.imread('test_image3.jpg')

sp_noisy_image = np.copy(image2)

num_salt = np.ceil(0.005 * image2.size * 0.5) # number of pixels that will be salt
num_pepper = np.ceil(0.005 * image2.size * 0.5) # number of pixels that will be pep

print(num_salt, num_pepper)
```

In [ ]:
```
# Add salt noise
coords = [np.random.randint(0, i - 1, int(num_salt)) for i in image2.shape]
sp_noisy_image[coords[0], coords[1], :] = 255 # the ':' means the specified coords

# Add pepper noise
```

```
coords = [np.random.randint(0, i - 1, int(num_pepper)) for i in image2.shape]
sp_noisy_image[coords[0], coords[1], :] = 0
```

In [ ]:
```
# Denoising using cv2 median blur function
sp_denoised_image = cv2.medianBlur(sp_noisy_image, 3)  # Apply median filter with k

# Denoising using median filter matrix
rows, cols, channels = sp_noisy_image.shape
sp_denoised_image2 = np.zeros((rows-2, cols-2, channels), dtype=np.uint8)
for i in range(1, rows - 1):
    for j in range(1, cols - 1):
        for k in range(channels):
            window = sp_noisy_image[i-1:i+2, j-1:j+2, k]
            sp_denoised_image2[i-1, j-1, k] = np.median(window)
```

In [ ]:
```
plt.figure(figsize=(5, 5))

plt.subplot(2, 2, 1)
plt.imshow(cv2.cvtColor(image2, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')

plt.subplot(2, 2, 2)
plt.imshow(cv2.cvtColor(sp_noisy_image, cv2.COLOR_BGR2RGB))
plt.title('Noisy Image')
plt.axis('off')

plt.subplot(2, 2, 3)
plt.imshow(cv2.cvtColor(sp_denoised_image, cv2.COLOR_BGR2RGB))
plt.title('Denoised Image')
plt.axis('off')

plt.subplot(2, 2, 4)
plt.imshow(cv2.cvtColor(sp_denoised_image2, cv2.COLOR_BGR2RGB))
plt.title('Denoised Image 2')
plt.axis('off')

plt.show()
```

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Artificial Intelligence & Machine Learning
Academic Year 2023-2024

Name: Abhay Mathur **SAPID: 60017210016**
Batch: A1

Experiment No. 4

Aim: Feature Detection in Images

Objective: Develop a program to detect features in an Image (Edge)

Theory:

Image feature extraction involves identifying and representing distinctive structures within an image. Reading the pixels of an image is certainly one. But this is a low-level feature. A high-level feature of an image can be anything from edges, corners, or even more complex textures and shapes.

Features are characteristics of an image. With these unique characteristics, you may be able to distinguish one image from another. This is the first step in computer vision. By extracting these features, you can create representations that are more compact and meaningful than merely the pixels of the image. It helps further analysis and processing.

An edge is defined as a gradient on the pixel intensity. In other words, if there is an abrupt color change, it is considered an edge

Sobel Mask

Following is the vertical Mask of Sobel Operator:

-1	0	1
-2	0	2
-1	0	1

This mask works exactly same as the Prewitt operator vertical mask. There is only one difference that is it has “2” and “-2” values in center of first and third column. When applied on an image this mask will highlight the vertical edges.

How it works

When we apply this mask on the image it prominent vertical edges. It simply works like as first order derivate and calculates the difference of pixel intensities in a edge region.

As the center column is of zero so it does not include the original values of an image but rather it calculates the difference of right and left pixel values around that edge. Also the center values of both the first and third column is 2 and -2 respectively.

This give more weight age to the pixel values around the edge region. This increase the edge intensity and it become enhanced comparatively to the original image.

Following is the horizontal Mask of Sobel Operator

-1	-2	-1
0	0	0
1	2	1

Above mask will find edges in horizontal direction and it is because that zeros column is in horizontal direction. When you will convolve this mask onto an image it would prominent horizontal edges in the image. The only difference between it is that it have 2 and -2 as a center element of first and third row.

How it works

This mask will prominent the horizontal edges in an image. It also works on the principle of above mask and calculates difference among the pixel intensities of a particular edge. As the center row of mask is consist of zeros so it does not include the original values of edge in the image but rather it calculate the difference of above and below pixel intensities of the particular edge. Thus increasing the sudden change of intensities and making the edge more visible.

Prewitt Mask

The Prewitt operator is used in image processing, particularly within edge detection algorithms. Technically, it is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Prewitt operator is either the corresponding gradient vector or the norm of this vector. The Prewitt operator is based on convolving the image with a small, separable, and

integer valued filter in horizontal and vertical directions and is therefore relatively inexpensive in terms of computations

Following is the vertical Mask of Prewitt Operator:

-1	0	1
-1	0	1
-1	0	1

Following is the horizontal Mask of Prewitt Operator

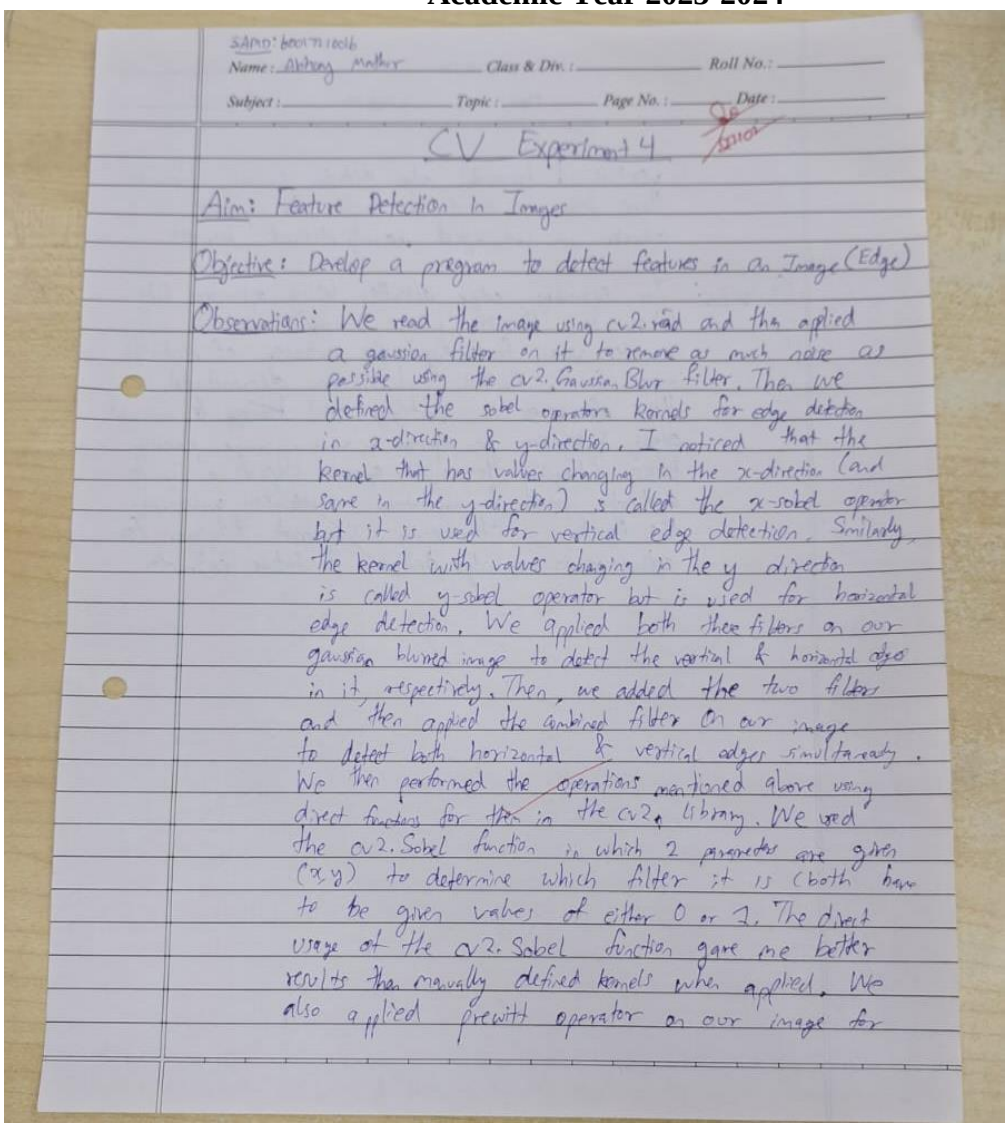
-1	-1	-1
0	0	0
1	1	1

Problem Definition

- Edge Detection using First Derivative Filters (Sobel, Prewitt) X direction, Y Direction, Both the directions
- Compare Results

Observations:

Department of Artificial Intelligence & Machine Learning
Academic Year 2023-2024



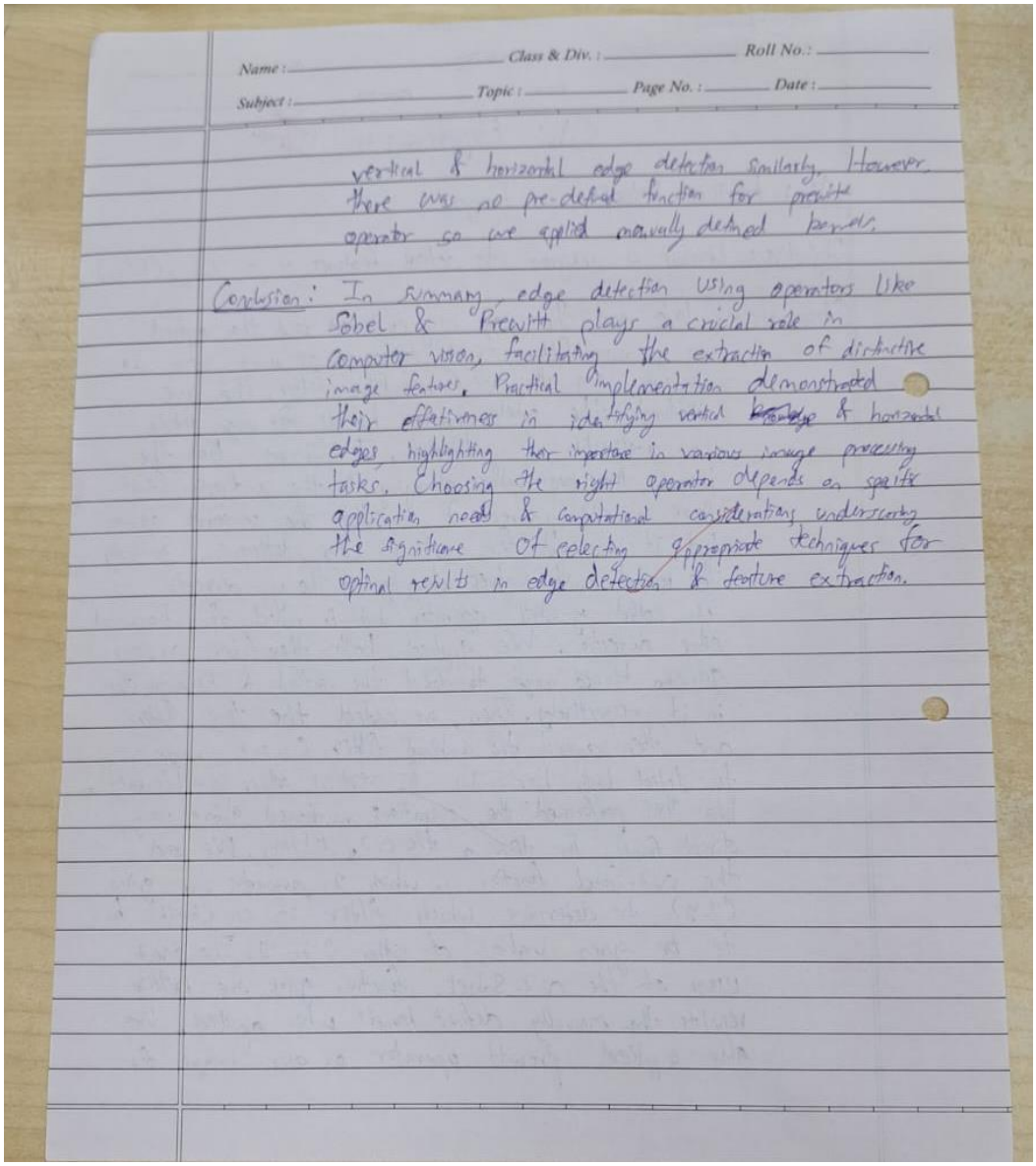
SAP ID: 60017 19016
Name: Akshay Mathur Class & Div.: Roll No.:
Subject: Topic: Page No.: Date:

CV Experiment 4

Aim: Feature Detection in Images

Objective: Develop a program to detect features in an Image (Edge)

Observations: We read the image using cv2.read and then applied a gaussian filter on it to remove as much noise as possible using the cv2.GaussianBlur filter. Then we defined the sobel operators kernels for edge detection in x-direction & y-direction. I noticed that the kernel that has values changing in the x-direction (and same in the y-direction) is called the x-sobel operator but it is used for vertical edge detection. Similarly, the kernel with values changing in the y direction is called y-sobel operator but is used for horizontal edge detection. We applied both these filters on our gaussian blurred image to detect the vertical & horizontal edges in it, respectively. Then, we added the two filters and then applied the combined filter on our image to detect both horizontal & vertical edges simultaneously. We then performed the operations mentioned above using direct functions for them in the cv2 library. We used the cv2.Sobel function in which 2 parameters are given (x,y) to determine which filter it is (both have to be given values of either 0 or 1. The direct usage of the cv2.Sobel function gave me better results than manually defined kernels when applied. We also applied prewitt operator on our image for



Name : Class & Div. : Roll No. :
Subject : Topic : Page No. : Date :

vertical & horizontal edge detection similarly. However there was no pre-defined function for prewitt operator so we applied manually defined kernels.

Conclusion: In summary, edge detection using operators like Sobel & Prewitt plays a crucial role in computer vision, facilitating the extraction of distinctive image features. Practical implementation demonstrated their effectiveness in identifying vertical & horizontal edges, highlighting their importance in various image processing tasks. Choosing the right operator depends on specific application needs & computational considerations, underscoring the significance of selecting appropriate techniques for optimal results in edge detection & feature extraction.

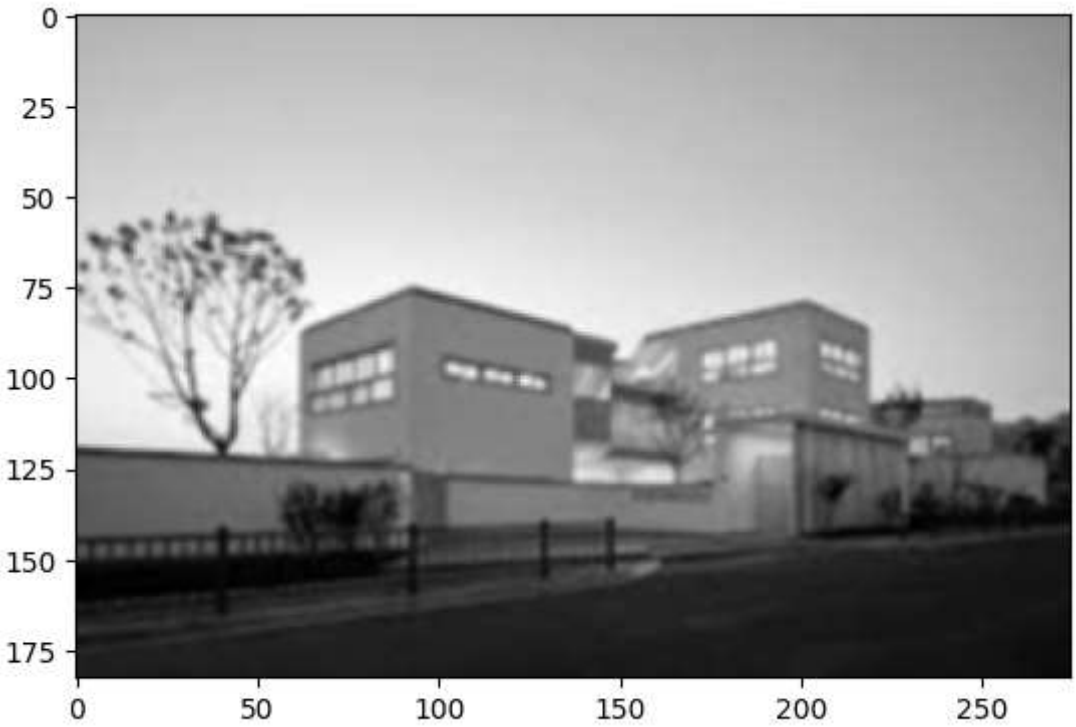
Conclusion:

In summary, edge detection using operators like Sobel and Prewitt plays a crucial role in computer vision, facilitating the extraction of distinctive image features. Practical implementation demonstrated their effectiveness in identifying vertical and horizontal edges, highlighting their importance in various image processing tasks. Choosing the right operator depends on specific application needs and computational considerations, underscoring the significance of selecting appropriate techniques for optimal results in edge detection and feature extraction.

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

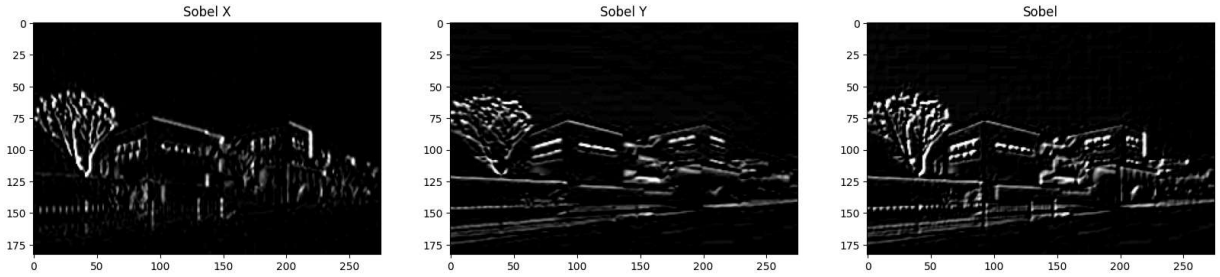
In [ ]:
```
img = cv2.imread('images.jpg',0)
plt.imshow(img, cmap='gray')
plt.show()
```

In [ ]:
```
img_gaussian = cv2.GaussianBlur(img,(3,3),0)
plt.imshow(img_gaussian, cmap='gray')
plt.show()
```

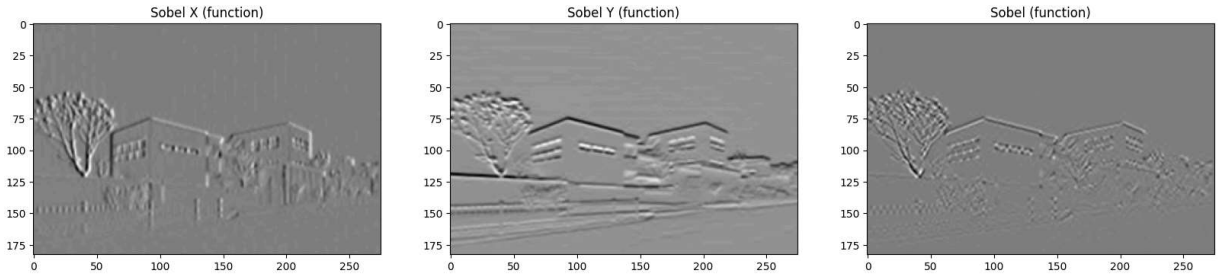


Sobel Operator

In [ ]:
```
Sobel_x = np.array([[-1, 0, 1],
                    [-2, 0, 2],
                    [-1, 0, 1]])
Sobel_y = np.array([[-1, -2, -1],
                    [0, 0, 0],
                    [1, 2, 1]])
sobel = Sobel_x + Sobel_y

img_sobelx = cv2.filter2D(img_gaussian, -1, Sobel_x)
img_sobely = cv2.filter2D(img_gaussian, -1, Sobel_y)
img_sobel = cv2.filter2D(img_gaussian, -1, sobel)

fig, ax = plt.subplots(1, 3, figsize=(20, 20))

ax[0].imshow(img_sobelx, cmap='gray')
ax[0].set_title('Sobel X')

ax[1].imshow(img_sobely, cmap='gray')
ax[1].set_title('Sobel Y')

ax[2].imshow(img_sobel, cmap='gray')
ax[2].set_title('Sobel')

plt.show()
```



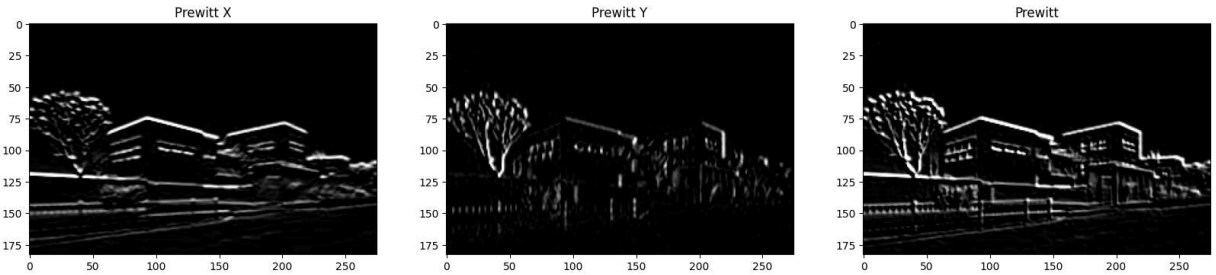
In [ ]:
```
sobel_x_function = cv2.Sobel(img_gaussian, cv2.CV_64F, 1,0, ksize=3)
sobel_y_function = cv2.Sobel(img_gaussian, cv2.CV_64F, 0,1, ksize=3)
sobel_xy_function = cv2.Sobel(img_gaussian, cv2.CV_64F, 1,1, ksize=3)

fig, ax = plt.subplots(1, 3, figsize=(20, 20))

ax[0].imshow(sobel_x_function, cmap='gray')
ax[0].set_title('Sobel X (function)')

ax[1].imshow(sobel_y_function, cmap='gray')
ax[1].set_title('Sobel Y (function)')

ax[2].imshow(sobel_xy_function, cmap='gray')
ax[2].set_title('Sobel (function)')

plt.show()
```



Prewitt Operator

In [ ]:
```
kernelx = np.array([[1,1,1],[0,0,0],[-1,-1,-1]])
kernely = np.array([[-1,0,1],[-1,0,1],[-1,0,1]])
kernel = kernelx + kernely

img_prewittx = cv2.filter2D(img_gaussian, -1, kernelx)
img_prewitty = cv2.filter2D(img_gaussian, -1, kernely)
img_prewitt = cv2.filter2D(img_gaussian, -1, kernel)

fig, ax = plt.subplots(1, 3, figsize=(20, 20))

ax[0].imshow(img_prewittx, cmap='gray')
ax[0].set_title('Prewitt X')

ax[1].imshow(img_prewitty, cmap='gray')
ax[1].set_title('Prewitt Y')

ax[2].imshow(img_prewitt, cmap='gray')
```

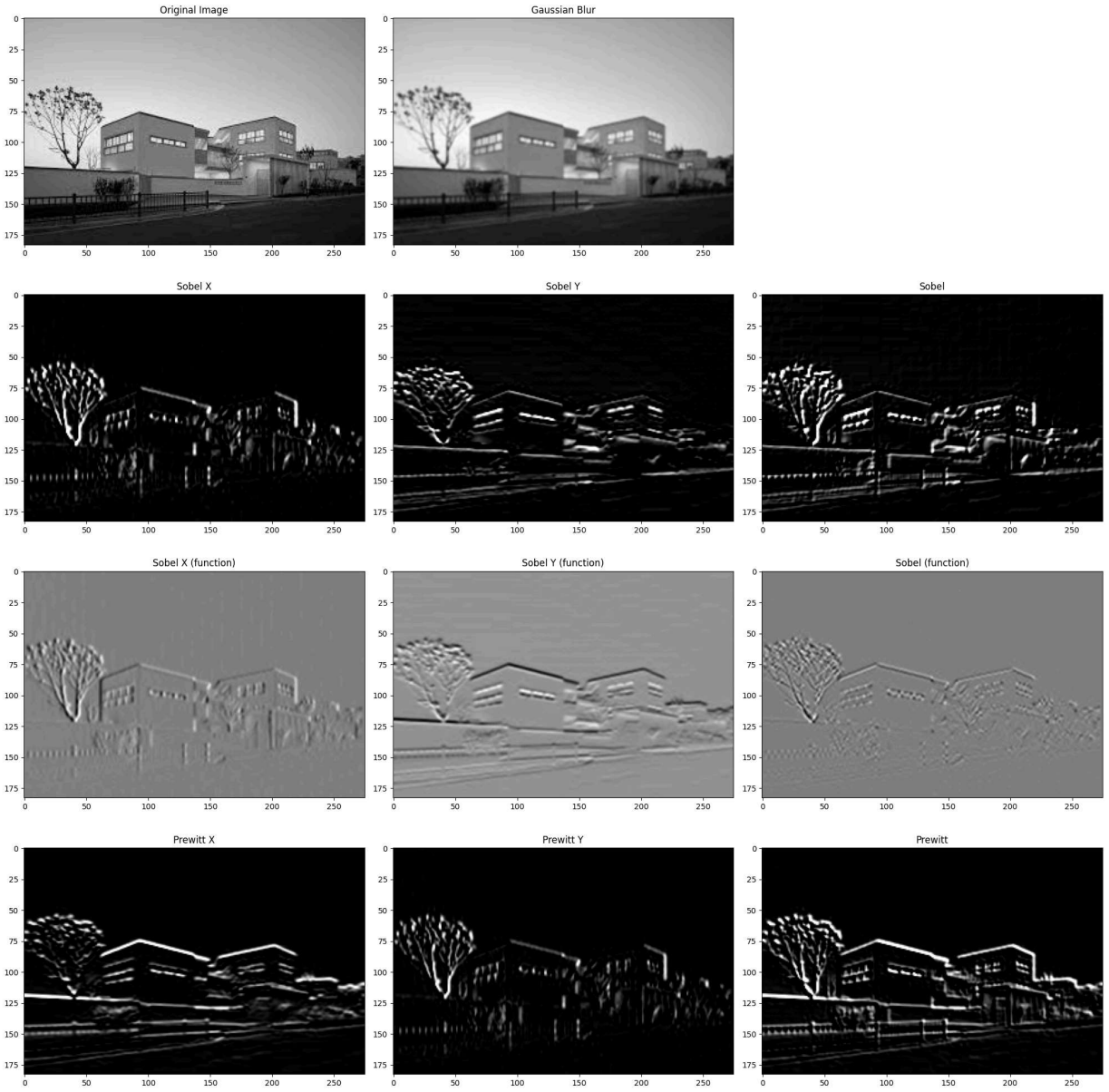
```
ax[2].set_title('Prewitt')

plt.show()
```



Display All:

In [ ]:
```
fig, ax = plt.subplots(4, 3, figsize=(20, 20))

# Cell 1
ax[0, 0].imshow(img, cmap='gray')
ax[0, 0].set_title('Original Image')

# Cell 2
ax[0, 1].imshow(img_gaussian, cmap='gray')
ax[0, 1].set_title('Gaussian Blur')

# Cell 3
plt.delaxes(ax[0, 2])

# Cell 4
ax[1, 0].imshow(img_sobelx, cmap='gray')
ax[1, 0].set_title('Sobel X')

# Cell 5
ax[1, 1].imshow(img_sobely, cmap='gray')
ax[1, 1].set_title('Sobel Y')

# Cell 6
ax[1, 2].imshow(img_sobel, cmap='gray')
ax[1, 2].set_title('Sobel')

# Cell 8
ax[2, 0].imshow(sobel_x_function, cmap='gray')
ax[2, 0].set_title('Sobel X (function)')

# Cell 9
ax[2, 1].imshow(sobel_y_function, cmap='gray')
ax[2, 1].set_title('Sobel Y (function)')

# Cell 10
ax[2, 2].imshow(sobel_xy_function, cmap='gray')
ax[2, 2].set_title('Sobel (function)')

# Cell 12
ax[3, 0].imshow(img_prewittx, cmap='gray')
ax[3, 0].set_title('Prewitt X')
```

```
# Cell 13
ax[3, 1].imshow(img_prewitty, cmap='gray')
ax[3, 1].set_title('Prewitt Y')

# Cell 14
ax[3, 2].imshow(img_prewitt, cmap='gray')
ax[3, 2].set_title('Prewitt')

plt.tight_layout()
plt.show()
```



Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Artificial Intelligence & Machine Learning
Academic Year 2023-2024

Name: Abhay Mathur **SAPID:** 60017210016
Batch: A1

Experiment No. 5A

Aim: Feature Detection in Images

Objective: Develop a program to detect features in an Image (Edge)

Theory:

Image feature extraction involves identifying and representing distinctive structures within an image. Reading the pixels of an image is certainly one. But this is a low-level feature. A high-level feature of an image can be anything from edges, corners, or even more complex textures and shapes.

Features are characteristics of an image. With these unique characteristics, you may be able to distinguish one image from another. This is the first step in computer vision. By extracting these features, you can create representations that are more compact and meaningful than merely the pixels of the image. It helps further analysis and processing.

An edge is defined as a gradient on the pixel intensity. In other words, if there is an abrupt color change, it is considered an edge

The Laplacian filter comes under the derivative filter category. It is a second-order filter used in image processing for edge detection and feature extraction.

Problem Definition

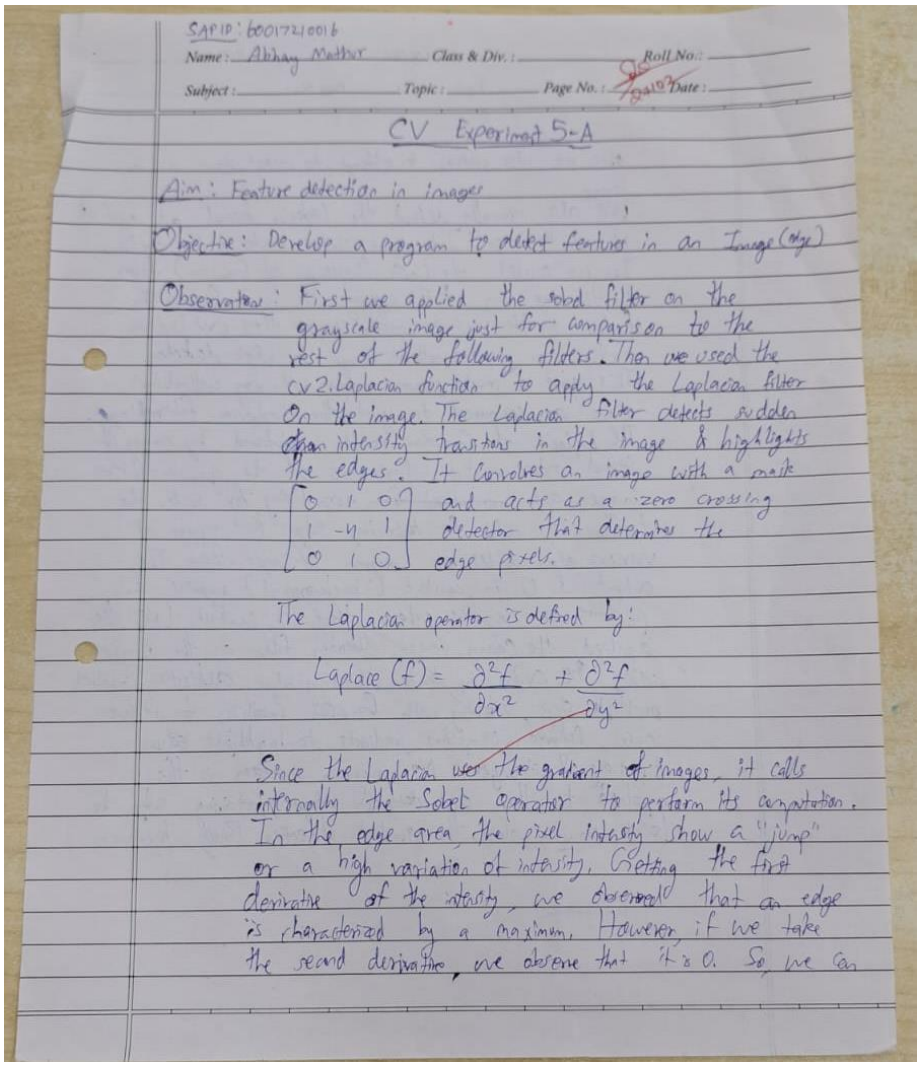
- Edge Detection (Sobel-x, Sobel-y, Sobel-Combined)
- Edge Detection using Laplacian
- Edge Detection using Laplacian of Gaussian
- Edge Detection using Canny Filter
- Compare Results

Post Lab Question

Explain Laplacian, Laplacian of Gaussian, Canny Edge Filter. Explain Canny Edge Filter Algorithm in detail

Department of Artificial Intelligence & Machine Learning
Academic Year 2023-2024

Observations



SAP ID: 60017210016
Name: Abhay Mathur Class & Div.: Roll No.:
Subject: Topic: Page No.: Date:

CV Experiment 5-A

Aim: Feature detection in images

Objective: Develop a program to detect features in an Image (edge)

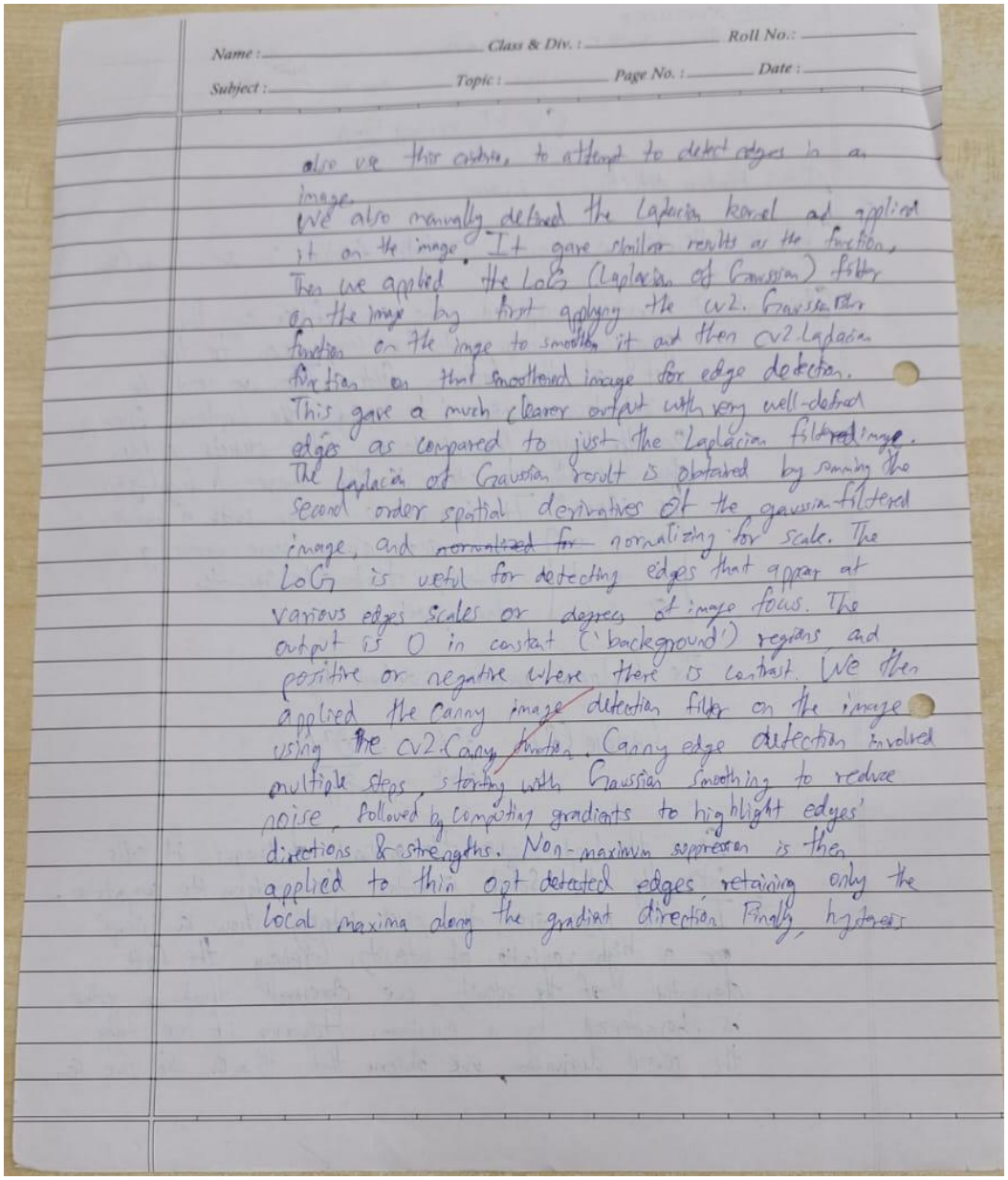
Observation: First we applied the sobel filter on the grayscale image just for comparison to the rest of the following filters. Then we used the cv2.Laplacian function to apply the Laplacian filter on the image. The Laplacian filter detects sudden intensity transitions in the image & highlights the edges. It convolves an image with a mask $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ and acts as a zero crossing detector that determines the edge pixels.

The Laplacian operator is defined by:

$$\text{Laplace}(f) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

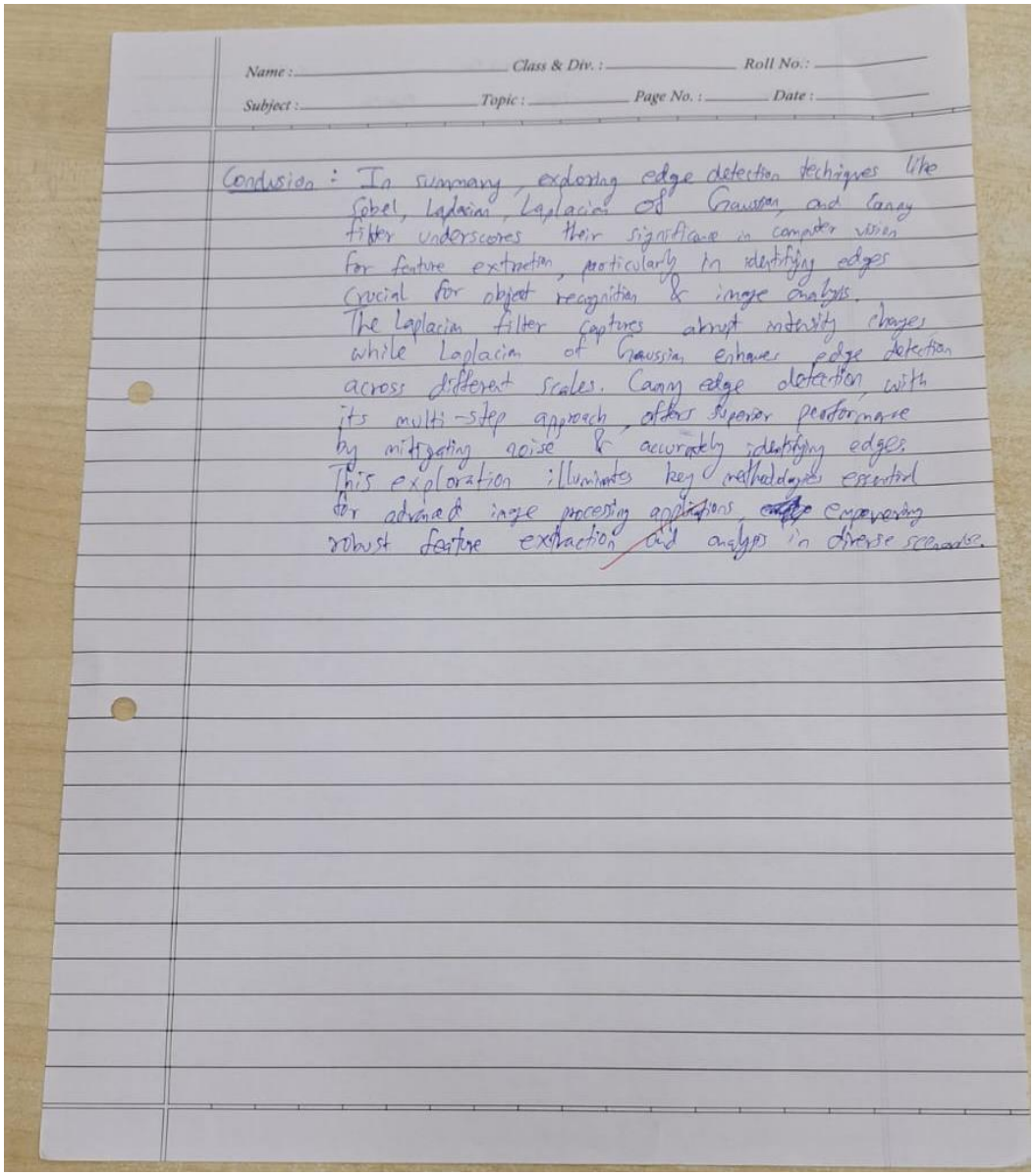
Since the Laplacian use the gradient of images, it calls internally the Sobel operator to perform its computation. In the edge area the pixel intensity show a "jump" or a high variation of intensity. Getting the first derivative of the intensity, we observed that an edge is characterized by a maximum. However if we take the second derivative, we observe that it's 0. So we can

Department of Artificial Intelligence & Machine Learning
Academic Year 2023-2024



Name : Class & Div. : Roll No.:
Subject : Topic : Page No. : Date :

also we this attribute, to attempt to detect edges in an image.
We also manually defined the Laplacian kernel and applied it on the image. It gave similar results as the function. Then we applied the LoG (Laplacian of Gaussian) filter on the image by first applying the cv2. Gaussian function on the image to smoothen it and then cv2 Laplacian function on that smoothened image for edge detection. This gave a much clearer output with very well-defined edges as compared to just the "Laplacian filtered image. The Laplacian of Gaussian result is obtained by applying the second order spatial derivatives of the gaussian-filtered image, and ~~normalized for~~ normalizing for scale. The LoG is useful for detecting edges that appear at various edges scales or degrees of image focus. The output is 0 in constant ('background') regions and positive or negative where there is contrast. We then applied the Canny image detection filter on the image using the cv2.Canny function. Canny edge detection involved multiple steps, starting with Gaussian smoothing to reduce noise, followed by computing gradients to highlight edges' directions & strengths. Non-maximum suppression is then applied to thin out detected edges, retaining only the local maxima along the gradient direction. Finally, hysteresis


Name : Class & Div. : Roll No.:
Subject : Topic : Page No. : Date :

Conclusion : In summary, exploring edge detection techiques like Sobel, Laplacian, Laplacian of Gaussian, and Canny filter underscores their significance in computer vision for feature extraction, particularly in identifying edges crucial for object recognition & image analysis. The Laplacian filter captures abrupt intensity changes while Laplacian of Gaussian enhances edge detection across different scales. Canny edge detection, with its multi-step approach, offers superior performance by mitigating noise & accurately identifying edges. This exploration illuminates key methodologies essential for advanced image processing applications, empowering robust feature extraction and analysis in diverse scenarios.

Conclusion

In summary, exploring edge detection techniques like Sobel, Laplacian, Laplacian of Gaussian, and Canny filter underscores their significance in computer vision for feature extraction, particularly in identifying edges crucial for object recognition and image analysis. The Laplacian filter captures abrupt intensity changes, while Laplacian of Gaussian enhances edge detection across different scales. Canny edge detection, with its multi-step approach, offers superior performance by mitigating noise and accurately identifying edges. This exploration illuminates key methodologies essential for advanced image processing applications, empowering robust feature extraction and analysis in diverse scenarios.

Sobel Operator (with function)

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

In [ ]:
```
image = cv2.imread('..\image.jpg', 0) # 0 for grayscale
sobel_xy = cv2.Sobel(image, cv2.CV_64F, 1,1, ksize=3)
```

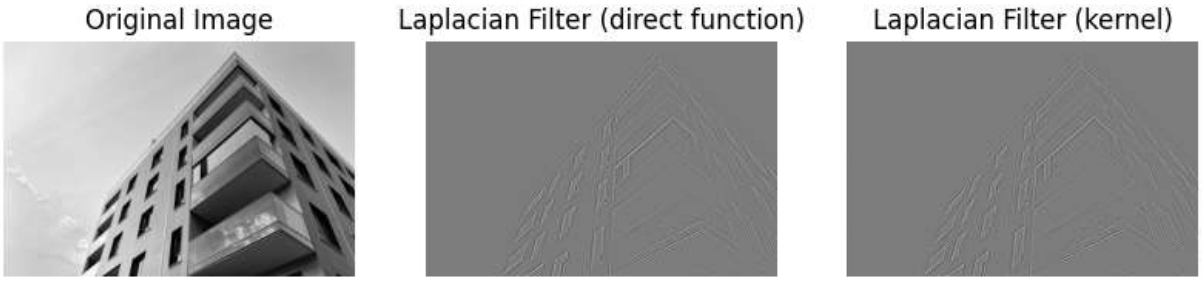
In [ ]:
```
fig, ax = plt.subplots(1, 2, figsize=(10, 5))
ax[0].imshow(image, cmap='gray')
ax[0].set_title('Original Image')
ax[0].axis('off')

ax[1].imshow(sobel_xy, cmap='gray')
ax[1].set_title('Sobel Filter')
ax[1].axis('off')

plt.show()
```

Laplacian Filter

In [ ]:
```
laplacian = cv2.Laplacian(image, cv2.CV_64F) # Using function


# Define the Laplacian kernel manually
laplacian_kernel = np.array([[0, 1, 0],
                             [1, -4, 1],
                             [0, 1, 0]])

# Apply the Laplacian filter
laplacian_filtered = cv2.filter2D(image, cv2.CV_64F, laplacian_kernel)
```

In [ ]:
```
fig, ax = plt.subplots(1, 3, figsize=(10, 5))
ax[0].imshow(image, cmap='gray')
ax[0].set_title('Original Image')
ax[0].axis('off')
```

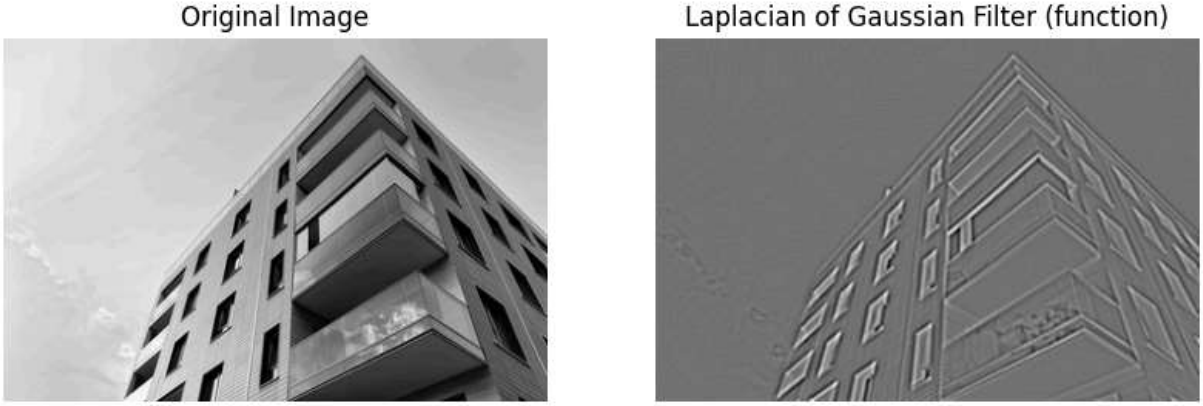
```
ax[1].imshow(laplacian, cmap='gray')
ax[1].set_title('Laplacian Filter (direct function)')
ax[1].axis('off')

ax[2].imshow(laplacian_filtered, cmap='gray')
ax[2].set_title('Laplacian Filter (kernel)')
ax[2].axis('off')


plt.show()
```



Laplacian of Gaussian (LoG)

In [ ]:
```
# Apply Gaussian blur
blurred_image = cv2.GaussianBlur(image, (5, 5), 0)  # You can adjust the kernel siz

# Apply Laplacian filter
laplacian_of_gaussian_using_function = cv2.Laplacian(blurred_image, cv2.CV_64F)
```

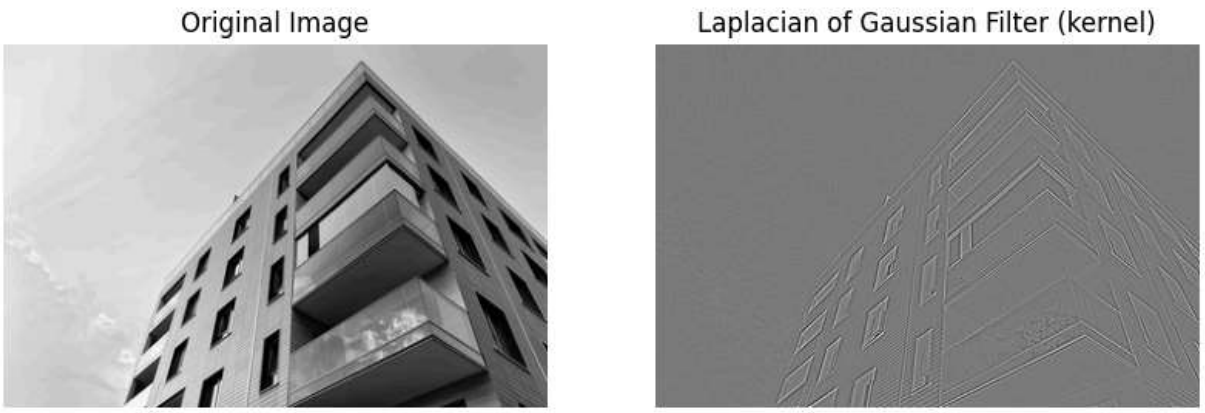
In [ ]:
```
fig, ax = plt.subplots(1, 2, figsize=(10, 5))
ax[0].imshow(image, cmap='gray')
ax[0].set_title('Original Image')
ax[0].axis('off')

ax[1].imshow(laplacian_of_gaussian_using_function, cmap='gray')
ax[1].set_title('Laplacian of Gaussian Filter (function)')
ax[1].axis('off')

plt.show()
```



In [ ]:
```
def laplacian_kernel(size):
    """
    Generates a Laplacian kernel of a given size.
    """
    kernel = np.zeros((size, size))
    center = size // 2
    for i in range(size):
        for j in range(size):
            kernel[i, j] = -(1 - ((i - center)**2 + (j - center)**2) / (2 * (size/2
    kernel /= np.sum(np.abs(kernel))
    return kernel
```

In [ ]:
```
# Apply Gaussian blur
blurred_image = cv2.GaussianBlur(image, (5, 5), 0)  # You can adjust the kernel siz

# Apply Laplacian filter
laplacian_kernel_2d = laplacian_kernel(5)  # Adjust the kernel size to match the Ga
laplacian_of_gaussian= cv2.filter2D(blurred_image, -1, laplacian_kernel_2d)
laplacian_of_gaussian_normalized = cv2.normalize(laplacian_filtered, None, 0, 255,
```

In [ ]:
```
fig, ax = plt.subplots(1, 2, figsize=(10, 5))
ax[0].imshow(image, cmap='gray')
ax[0].set_title('Original Image')
ax[0].axis('off')

ax[1].imshow(laplacian_of_gaussian_normalized, cmap='gray')
ax[1].set_title('Laplacian of Gaussian Filter (kernel)')
ax[1].axis('off')

plt.show()
```

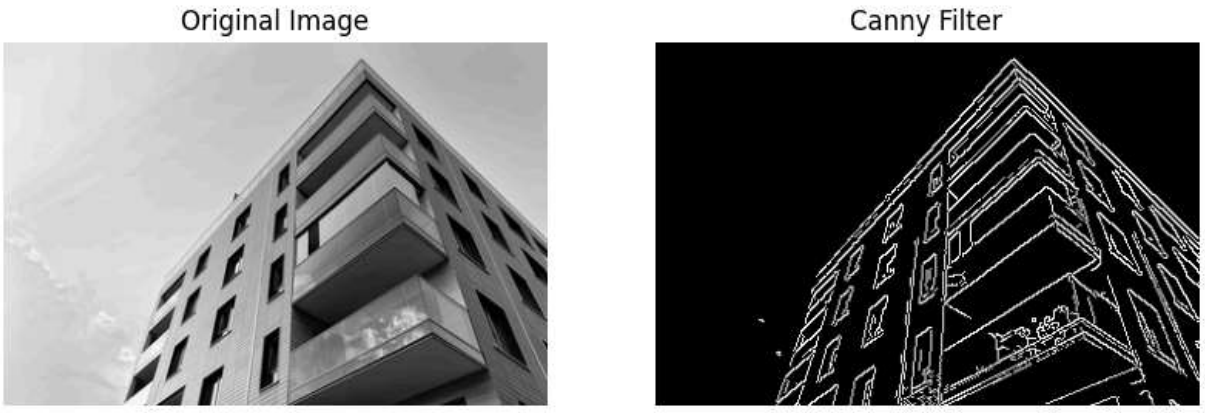


Canny Filter

In [ ]:
```
canny = cv2.Canny(image, 100, 200)
```

In [ ]:
```
fig, ax = plt.subplots(1, 2, figsize=(10, 5))
ax[0].imshow(image, cmap='gray')
ax[0].set_title('Original Image')
ax[0].axis('off')
```

```
ax[1].imshow(canny, cmap='gray')
ax[1].set_title('Canny Filter')
ax[1].axis('off')

plt.show()
```

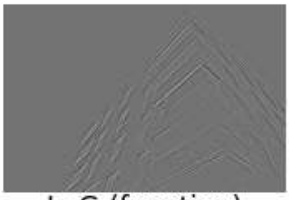


Display all:

In [ ]:
```
fig, ax = plt.subplots(3,3, figsize=(10, 5))
ax[0][0].imshow(image, cmap='gray')
ax[0][0].set_title('Original Image')
ax[0][0].axis('off')

ax[0][1].imshow(sobel_xy, cmap='gray')
ax[0][1].set_title('Sobel Filter')
ax[0][1].axis('off')

ax[0][2].imshow(laplacian, cmap='gray')
ax[0][2].set_title('Laplacian (function)')
ax[0][2].axis('off')

ax[1][0].imshow(laplacian_filtered, cmap='gray')
ax[1][0].set_title('Laplacian (kernel)')
ax[1][0].axis('off')

ax[1][1].imshow(laplacian_of_gaussian_using_function, cmap='gray')
ax[1][1].set_title('LoG (function)')
ax[1][1].axis('off')

ax[1][2].imshow(laplacian_of_gaussian_normalized, cmap='gray')
ax[1][2].set_title('LoG (kernel)')
ax[1][2].axis('off')

ax[2][0].imshow(canny, cmap='gray')
ax[2][0].set_title('Canny Filter')
ax[2][0].axis('off')

fig.delaxes(ax[2][1])
fig.delaxes(ax[2][2])

# plt.tight_layout()
plt.show()
```

Original Image

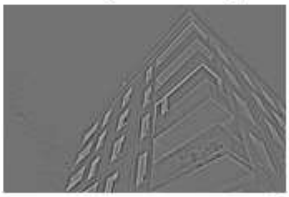
Sobel Filter



Laplacian (function)

Laplacian (kernel)

LoG (function)



LoG (kernel)

Canny Filter

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Artificial Intelligence & Machine Learning
Academic Year 2023-2024

Name: Abhay Mathur **SAPID:** 60017210016
Batch: A1

Experiment No. 5B

Aim: Feature Detection in Images

Objective: Develop a program to detect features in an Image (Edge)

Theory:

Difference of Gaussians (DoG) is calculated as the difference between two smoothed versions of an image obtained by applying two Gaussian kernels of different standard deviations (sigma) on that image

As an image enhancement algorithm, the difference of Gaussians can be utilized to increase the visibility of edges and other detail present in a digital image.

A wide variety of alternative edge sharpening filters operate by enhancing high frequency detail, but because random noise also has a high spatial frequency, many of these sharpening filters tend to enhance noise, an undesirable artifact.

The difference of Gaussians algorithm removes high frequency detail that often includes random noise, rendering this approach one of the most suitable for processing images with a high degree of noise.

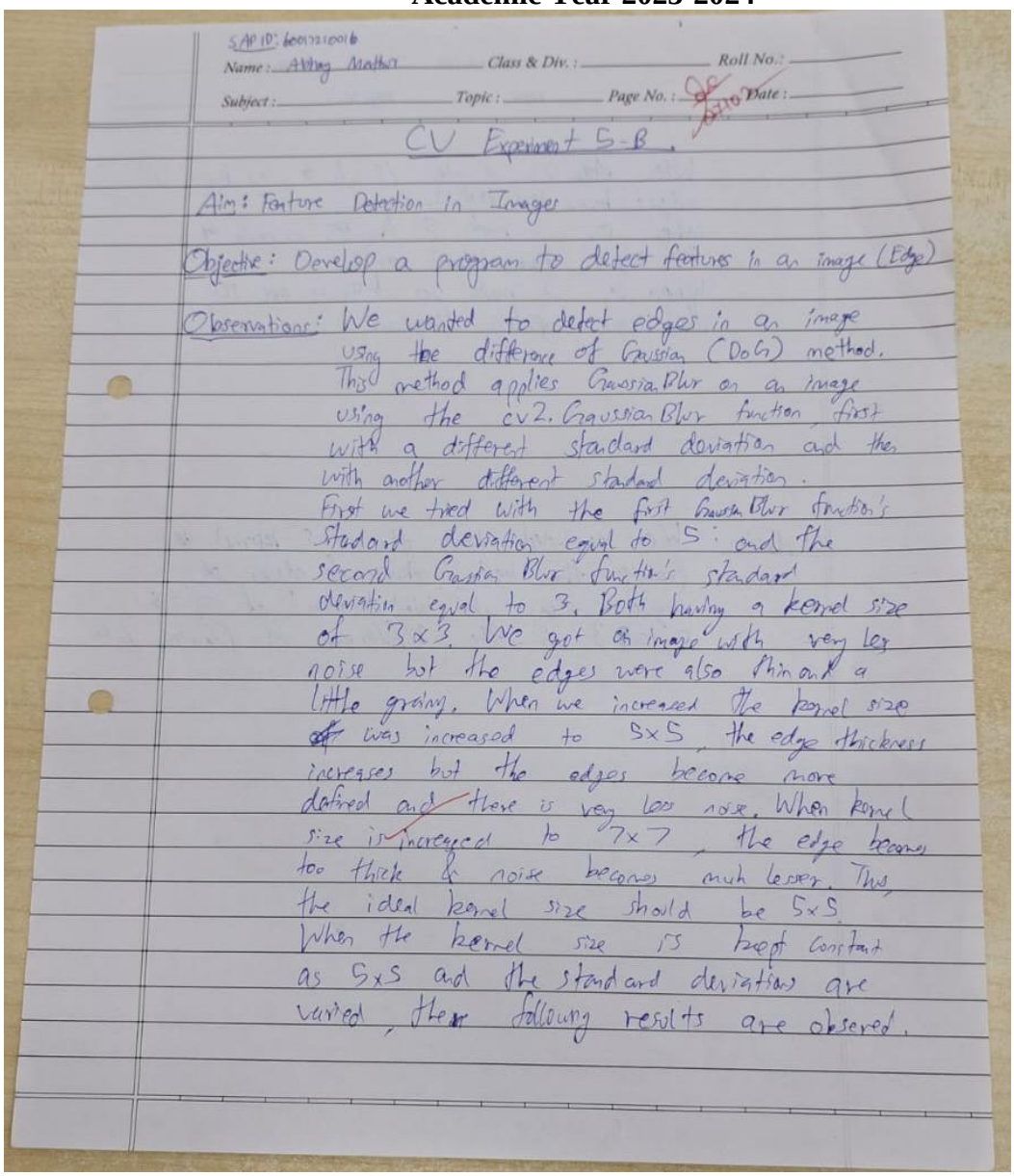
A major drawback to application of the difference of Gaussians algorithm is an inherent reduction in overall image contrast produced by the operation.

Problem Definition

- Edge Detection using Difference of Gaussian

Observations

Department of Artificial Intelligence & Machine Learning
Academic Year 2023-2024



SAP ID: 60017210016
Name: Aditya Mathur Class & Div.: Roll No.:
Subject: Topic: Page No.: Date:

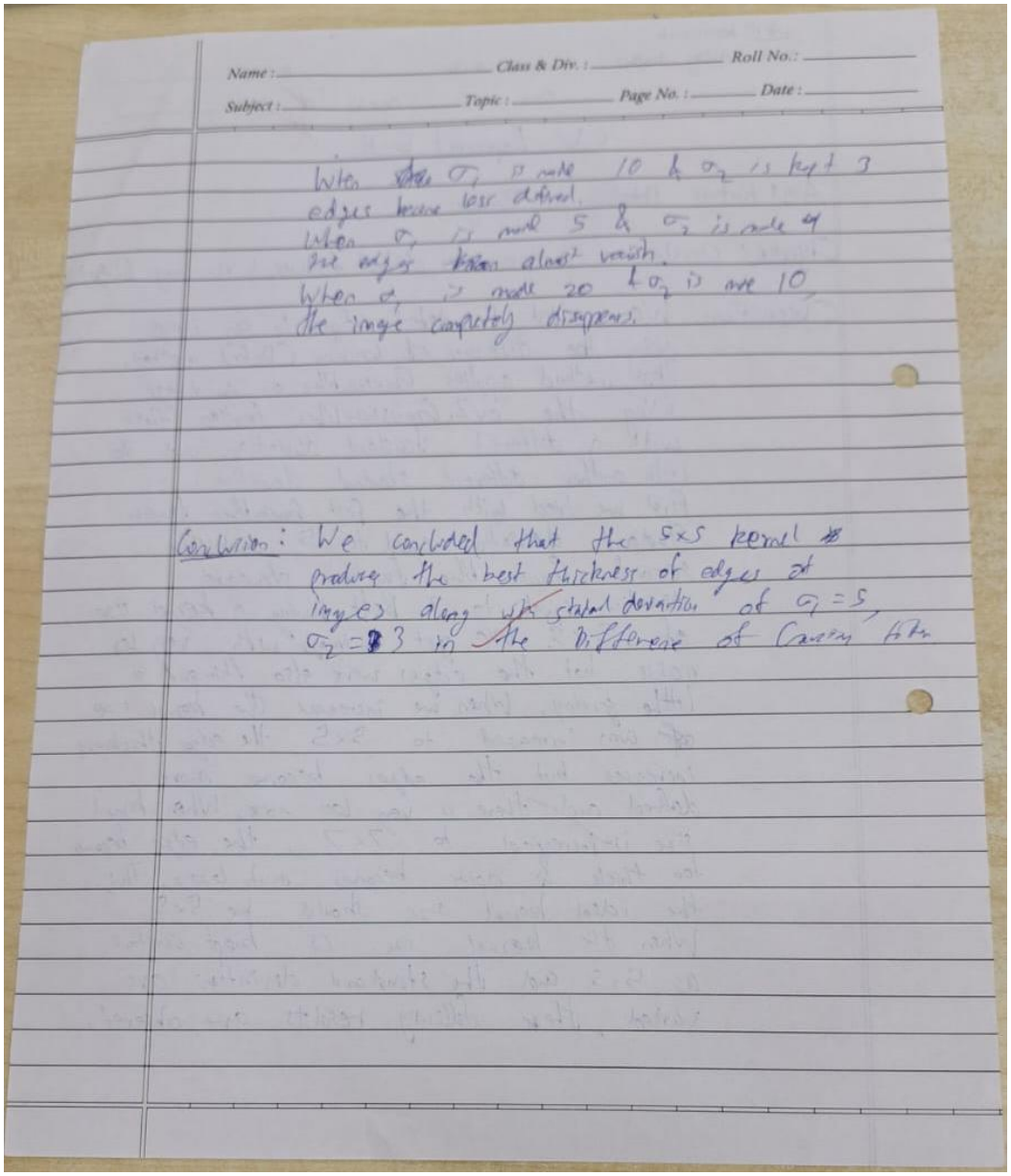
CV Experiment 5-B.

Aim: Feature Detection in Images

Objective: Develop a program to detect features in an image (Edge)

Observations: We wanted to detect edges in an image using the difference of Gaussian (DoG) method. This method applies Gaussian Blur on an image using the cv2.GaussianBlur function, first with a different standard deviation and then with another different standard deviation. First we tried with the first GaussianBlur function's standard deviation equal to 5 and the second GaussianBlur function's standard deviation equal to 3, both having a kernel size of 3x3. We got an image with very less noise but the edges were also thinned a little grainy. When we increased the kernel size ~~of~~ was increased to 5x5, the edge thickness increases but the edges become more defined and there is very less noise. When kernel size is increased to 7x7, the edge becomes too thick & noise becomes much lesser. Thus the ideal kernel size should be 5x5. When the kernel size is kept constant as 5x5 and the standard deviations are varied, these following results are observed.

Department of Artificial Intelligence & Machine Learning
Academic Year 2023-2024



Name : Class & Div. : Roll No. :
Subject : Topic : Page No. : Date :

When σ_1 is made 10 & σ_2 is kept 3 edges become less defined.
When σ_1 is made 5 & σ_2 is made 4 the edges become almost vanish.
When σ_1 is made 20 & σ_2 is made 10, the image completely disappears.

Conclusion: We concluded that the 5x5 kernal produce the best thickness of edges of images along with standard deviation of $\sigma_1 = 5$, $\sigma_2 = 3$ in the Difference of Gaussian filter

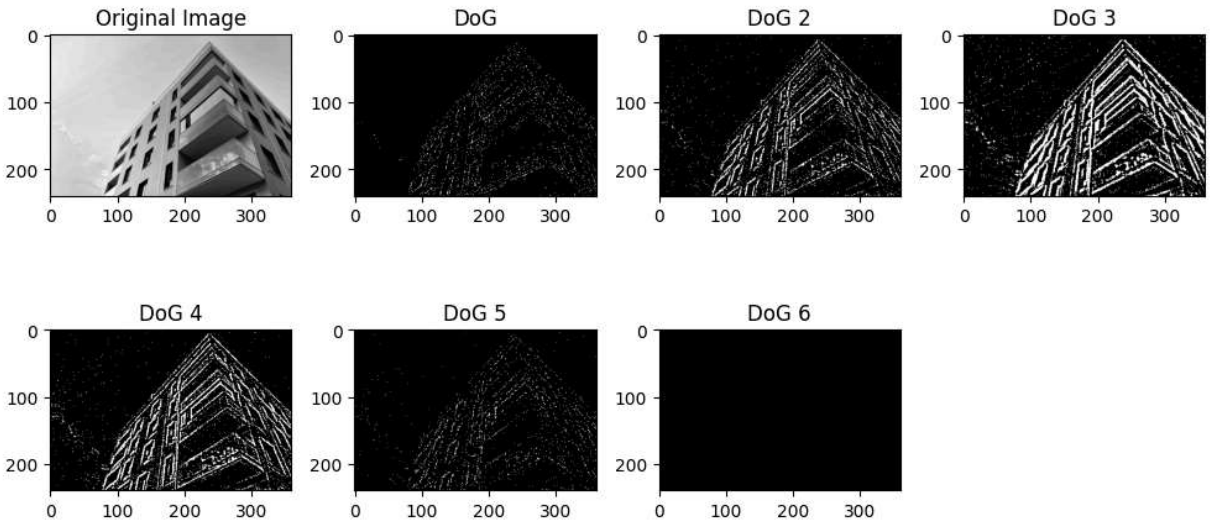
In [ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```
def difference_of_gaussians(image, sigma_large, sigma_small, kernel_size1, kernel_s
    # Apply Gaussian blur with large sigma
    gaussian_large = cv2.GaussianBlur(image, kernel_size1, sigma_large)

    # Apply Gaussian blur with small sigma
    gaussian_small = cv2.GaussianBlur(image, kernel_size2, sigma_small)

    # Calculate the difference of the two blurred images
    dog = gaussian_large - gaussian_small

    return dog
```

In [ ]:
```
image = cv2.imread('..\image.jpg',0)
plt.imshow(image, cmap='gray')
plt.show()
```

In [ ]:
```
dog_image = difference_of_gaussians(image, 5, 3, (3, 3), (3, 3))
dog_image2 = difference_of_gaussians(image, 5, 3, (5, 5), (5, 5))
dog_image3 = difference_of_gaussians(image, 5, 3, (7, 7), (7, 7))
dog_image4 = difference_of_gaussians(image, 10, 3, (5, 5), (5, 5))
dog_image5 = difference_of_gaussians(image, 5, 4, (5, 5), (5, 5))
dog_image6 = difference_of_gaussians(image, 20, 10, (5, 5), (5, 5))



fig, ax = plt.subplots(2, 4, figsize=(10, 5))
```

```
ax[0][0].imshow(image, cmap='gray')
ax[0][0].set_title('Original Image')

ax[0][1].imshow(dog_image, cmap='gray')
ax[0][1].set_title('DoG')

ax[0][2].imshow(dog_image2, cmap='gray')
ax[0][2].set_title('DoG 2')

ax[0][3].imshow(dog_image3, cmap='gray')
ax[0][3].set_title('DoG 3')

ax[1][0].imshow(dog_image4, cmap='gray')
ax[1][0].set_title('DoG 4')

ax[1][1].imshow(dog_image5, cmap='gray')
ax[1][1].set_title('DoG 5')

ax[1][2].imshow(dog_image6, cmap='gray')
ax[1][2].set_title('DoG 6')

fig.delaxes(ax[1][3])


plt.tight_layout()
plt.show()
```



Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Artificial Intelligence & Machine Learning
Academic Year 2023-2024

Name: Abhay Mathur **SAPID:** 60017210016
Batch: A1

Experiment No. 5C

Aim: Feature Detection in Images

Objective: Develop a program to detect features in an Image (Corner)

Theory:

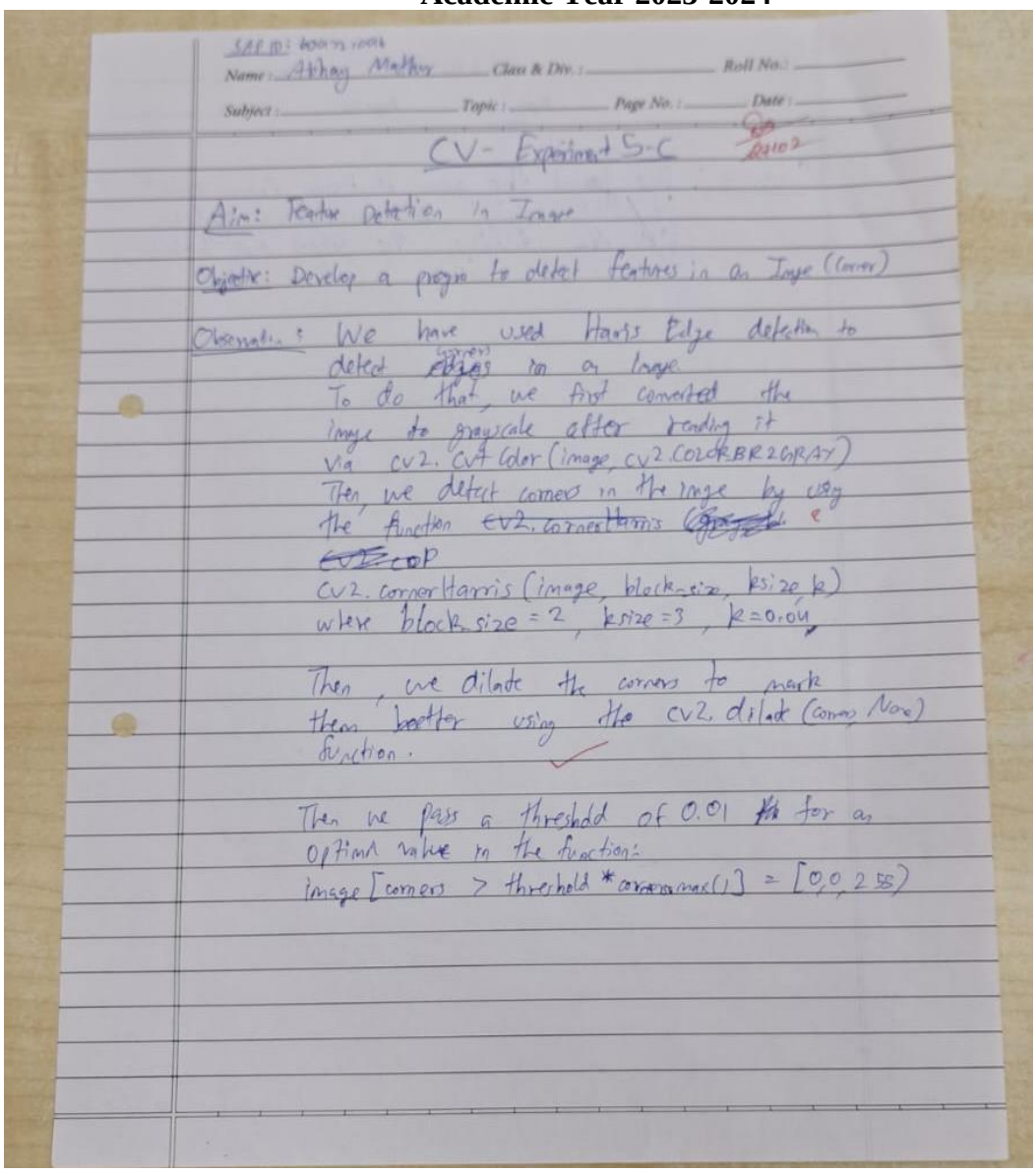
Harris Corner detection algorithm was developed to identify the internal corners of an image. The corners of an image are basically identified as the regions in which there are variations in large intensity of the gradient in all possible dimensions and directions. Corners extracted can be a part of the image features, which can be matched with features of other images, and can be used to extract accurate information. Harris Corner Detection is a method to extract the corners from the input image and to extract features from the input image

Problem Definition

- Corner Detection using Harris Operator

Observations

Department of Artificial Intelligence & Machine Learning
Academic Year 2023-2024



SAP ID: 60009210081
Name: Athang Mathur Class & Div.: Roll No.:
Subject: Topic: Page No.: Date:

CV- Experiment 5-C

Aim: Feature detection in Image

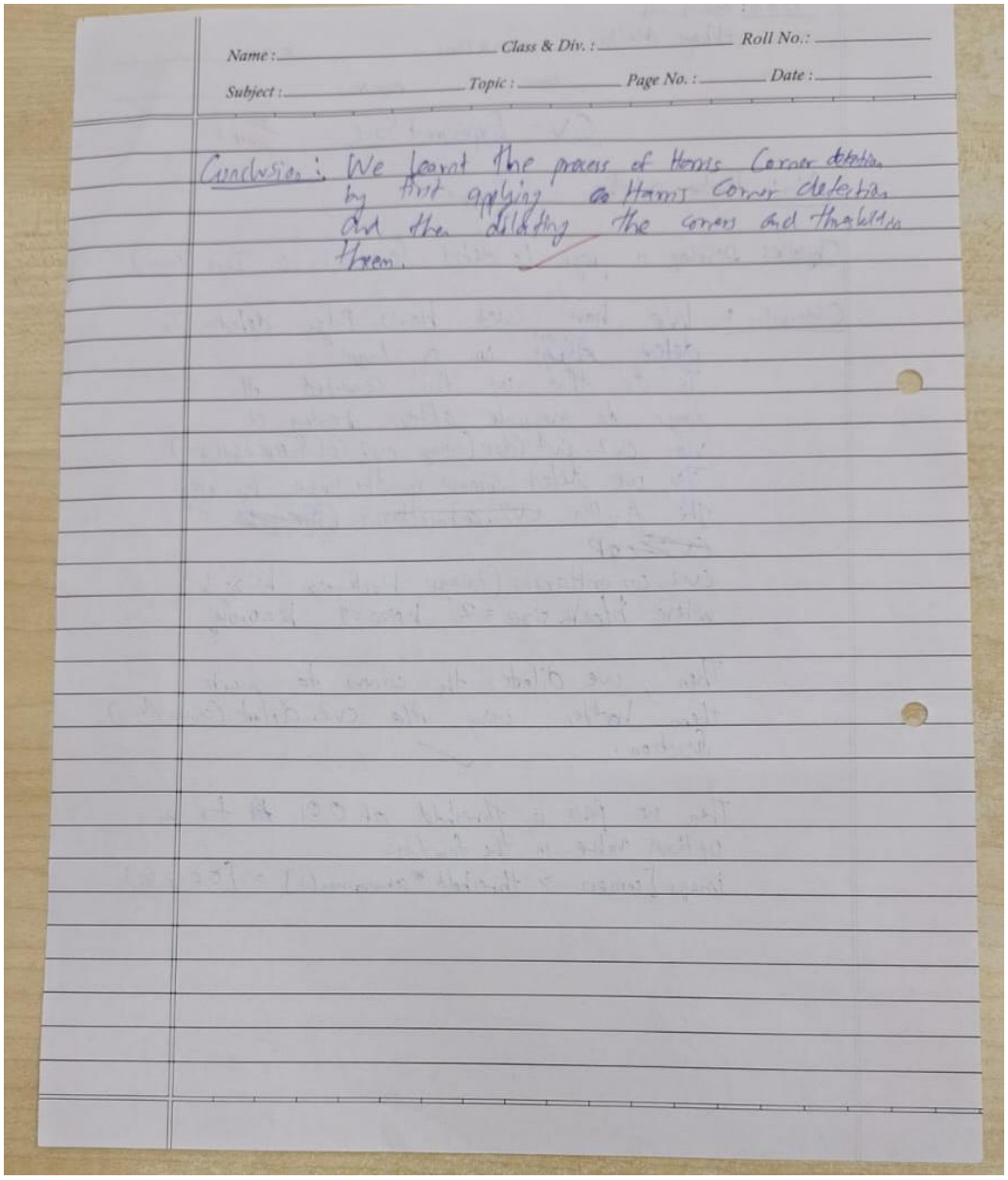
Objective: Develop a program to detect features in an Image (Corner)

Observation: We have used Harris Edge detection to detect corners in an image.
To do that, we first converted the image to grayscale after reading it via cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
Then, we detect corners in the image by using the function cv2.cornerHarris()
cv2.cornerHarris(image, block_size, ksize, k)
where block_size = 2, ksize = 3, k = 0.04

Then, we dilate the corners to mark them better using the cv2.dilate(corners, None) function.

Then we pass a threshold of 0.01 for an optimal value in the function:
image[corners > threshold * corners.max()] = [0, 0, 255]

Department of Artificial Intelligence & Machine Learning
Academic Year 2023-2024


Name : Class & Div. : Roll No. :
Subject : Topic : Page No. : Date :

Conclusion : We learnt the process of Harris Corner detection by first applying the Harris Corner detection and then plotting the corners and thresholding them.

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

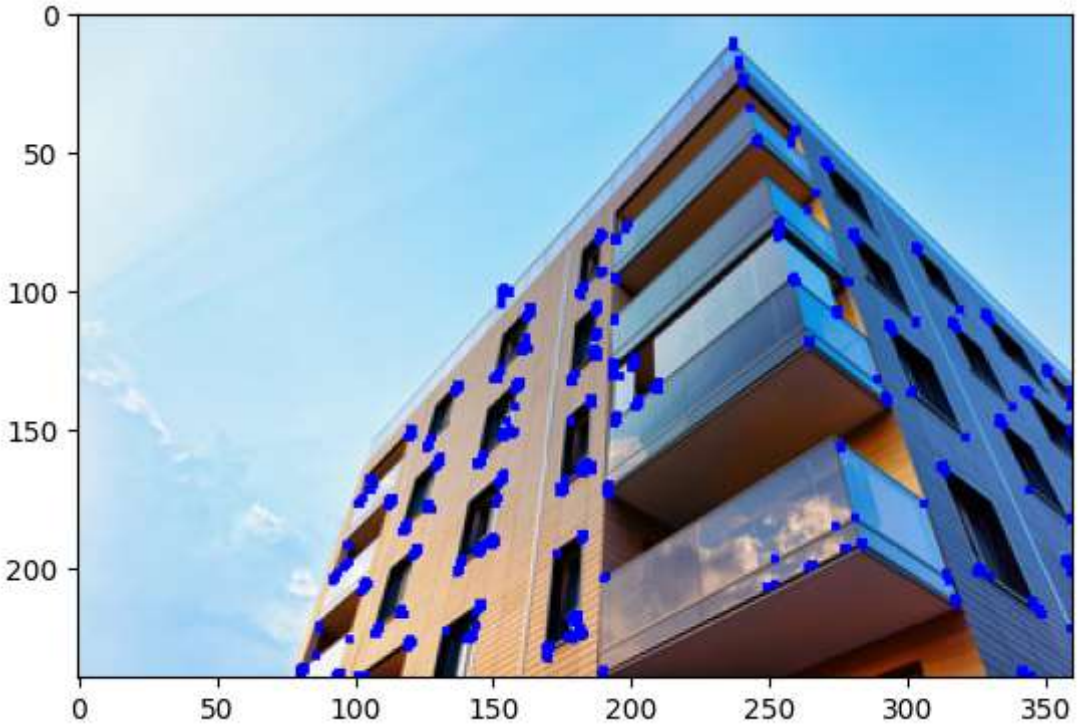
In [ ]:
```
def harris_edge_detection(image, block_size=2, ksize=3, k=0.04, threshold=0.01):
    # Convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Apply Harris corner detection
    corners = cv2.cornerHarris(gray, block_size, ksize, k)

    # Dilate corners to mark them better
    corners = cv2.dilate(corners, None)

    # Threshold for an optimal value, it may vary depending on the image
    image[corners > threshold * corners.max()] = [0, 0, 255]  # Mark detected corne

    return image
```

In [ ]:
```
image = cv2.imread('..\image.jpg')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(image)
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x2541dabb650>
```

In [ ]:
```
result = harris_edge_detection(image)
plt.imshow(result,cmap="gray")
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x2541da0aad0>
```


0
50
100
150
200
0
50
100
150
200
250
300
350

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Artificial Intelligence & Machine Learning
Academic Year 2023-2024

Name: Abhay Mathur **SAPID:** 60017210016
Batch: A1

Experiment No. 6

Aim: Object Detection

Objective: Develop a program to detect objects in an Image

Theory:

Object Detection is a computer technology related to computer vision, image processing, and deep learning that deals with detecting instances of objects in images and videos

SIFT stands for Scale-Invariant Feature Transform and was first presented in 2004, by **D.Lowe**, University of British Columbia. SIFT is invariance to image scale and rotation

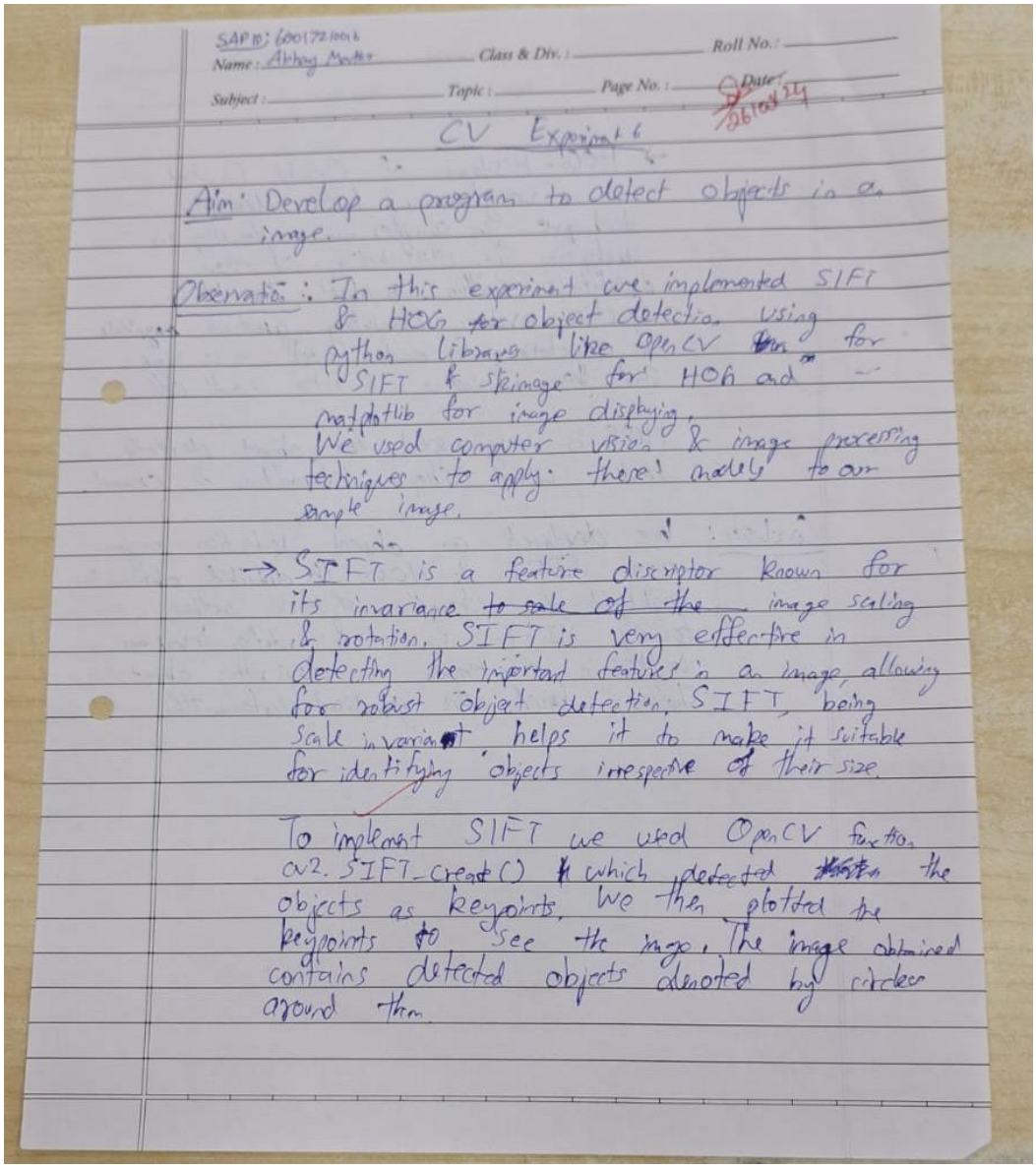
The Histogram of Oriented Gradients (HOG) is a popular feature descriptor technique in computer vision and image processing. It analyzes the distribution of edge orientations within an object to describe its shape and appearance. The HOG method involves computing the gradient magnitude and orientation for each pixel in an image and then dividing the image into small cells.

Problem Definition

- Object Detection in an image using SIFT
- Object Detection in an image using HOG

Observations

Department of Artificial Intelligence & Machine Learning
Academic Year 2023-2024



SAP ID: 60017210016
Name: Abhay Mehta Class & Div.: Roll No.:
Subject: Topic: Page No.: Date: 26/04/24

CV Experiment 6

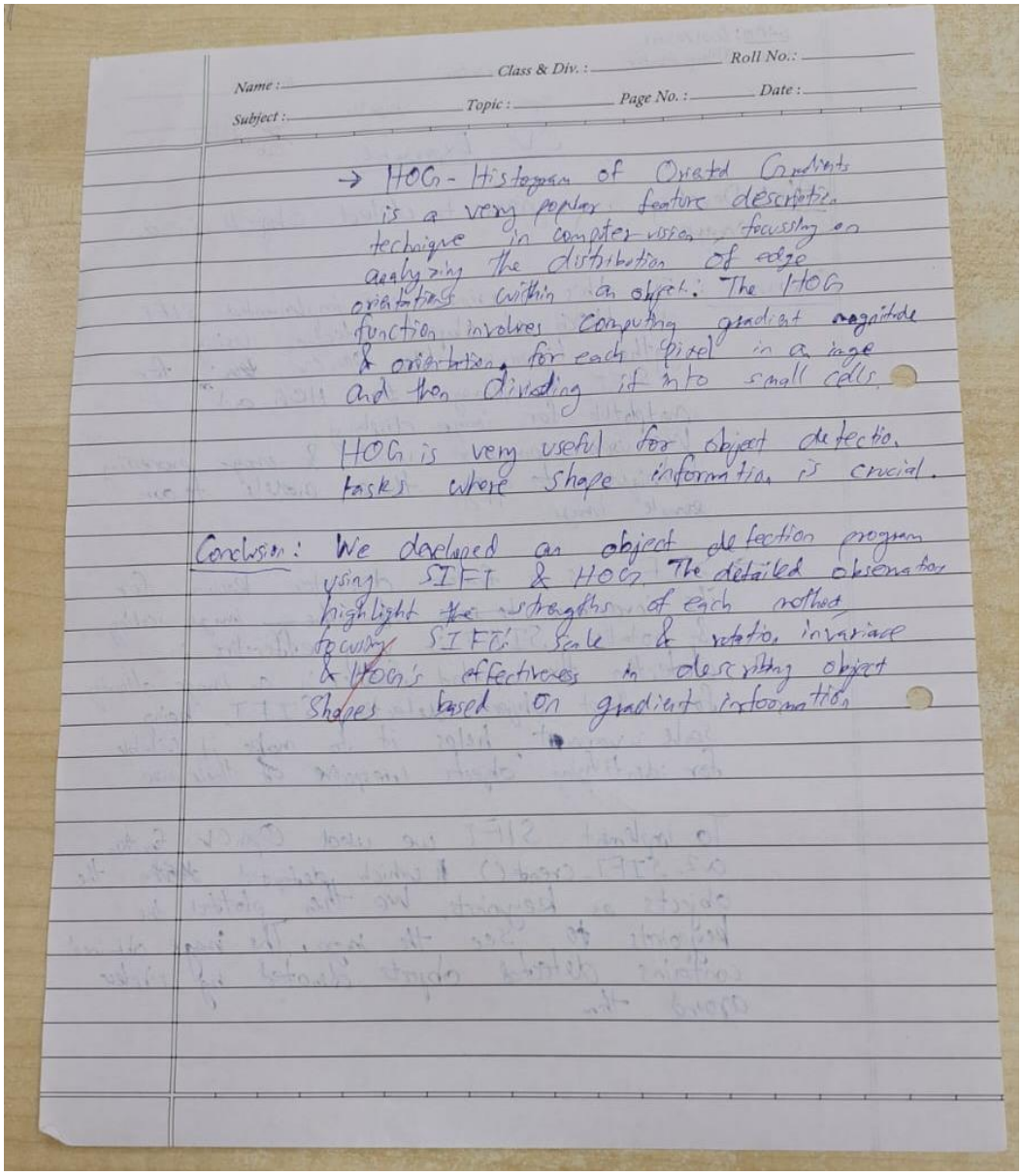
Aim: Develop a program to detect objects in a image.

Observation: In this experiment we implemented SIFT & HOG for object detection using python libraries like OpenCV for SIFT & Skimage for HOG and matplotlib for image displaying. We used computer vision & image processing techniques to apply these models to an sample image.

→ SIFT is a feature descriptor known for its invariance to scale of the image scaling & rotation. SIFT is very effective in detecting the important features in an image, allowing for robust object detection. SIFT, being scale invariant helps it to make it suitable for identifying objects irrespective of their size.

To implement SIFT we used OpenCV function cv2.SIFT_create() which detected the objects as keypoints. We then plotted the keypoints to see the image. The image obtained contains detected objects denoted by circles around them.

Department of Artificial Intelligence & Machine Learning
Academic Year 2023-2024



Name : Class & Div. : Roll No.:
Subject : Topic : Page No. : Date :

→ HOG - Histogram of Oriented Gradients is a very popular feature descriptor technique in computer vision, focussing on analyzing the distribution of edge orientations within an object. The HOG function involves computing gradient magnitude & orientation for each pixel in a image and then dividing it into small cells.

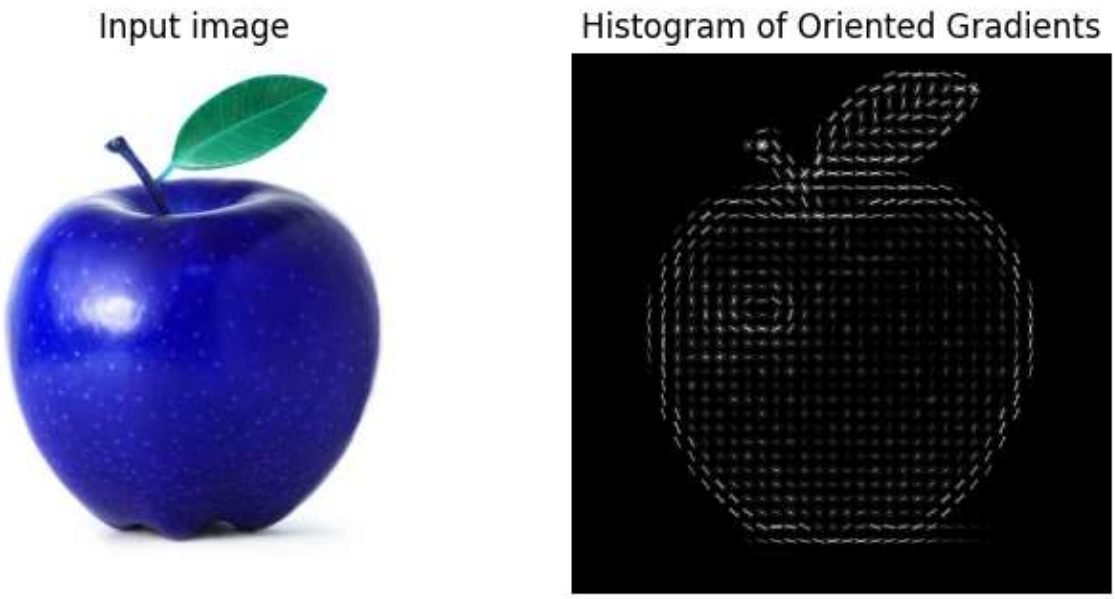
HOG is very useful for object detection tasks where shape information is crucial.

Conclusion: We developed an object detection program using SIFT & HOG. The detailed observations highlight the strengths of each method focusing SIFT's scale & rotation invariance & HOG's effectiveness in describing object shapes based on gradient information

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

# Loading the image
img = cv2.imread('test1.png')

 # Converting image to grayscale
gray= cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

# Applying SIFT detector
sift = cv2.SIFT_create()
kp = sift.detect(gray, None)

# Marking the keypoint on the image using circles
img=cv2.drawKeypoints(gray ,
                      kp ,
                      img ,
                      flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

cv2.imwrite('image-with-keypoints.jpg', img)
```

Out[ ]: True

In [ ]:
```
import matplotlib.pyplot as plt

from skimage.feature import hog
from skimage import data, exposure


image = cv2.imread('test2.jpg')

fd, hog_image = hog(image, orientations=8, pixels_per_cell=(16, 16),
                    cells_per_block=(1, 1), visualize=True, channel_axis=-1)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=True)

ax1.axis('off')
ax1.imshow(image, cmap=plt.cm.gray)
ax1.set_title('Input image')

# Rescale histogram for better display
hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10))

ax2.axis('off')
ax2.imshow(hog_image_rescaled, cmap=plt.cm.gray)
ax2.set_title('Histogram of Oriented Gradients')
plt.show()
```



In [ ]:

Department of Artificial Intelligence & Machine Learning

Academic Year 2023-2024

Name: Abhay Mathur **SAPID:** 60017210016
Batch: A1

Experiment No. 7

Aim: Image Segmentation

Objective: Develop a program to Segment Image using K Means Algorithm

Theory:

Image segmentation is the task of partitioning an image into multiple segments. In semantic segmentation, all pixels that are part of the same object type get assigned to the same segment.

Image segmentation is the process of partitioning a digital image into multiple distinct regions containing each pixel (sets of pixels, also known as superpixels) with similar attributes.

The goal of Image segmentation is to change the representation of an image into something that is more meaningful and easier to analyze.

Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, Image Segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics.

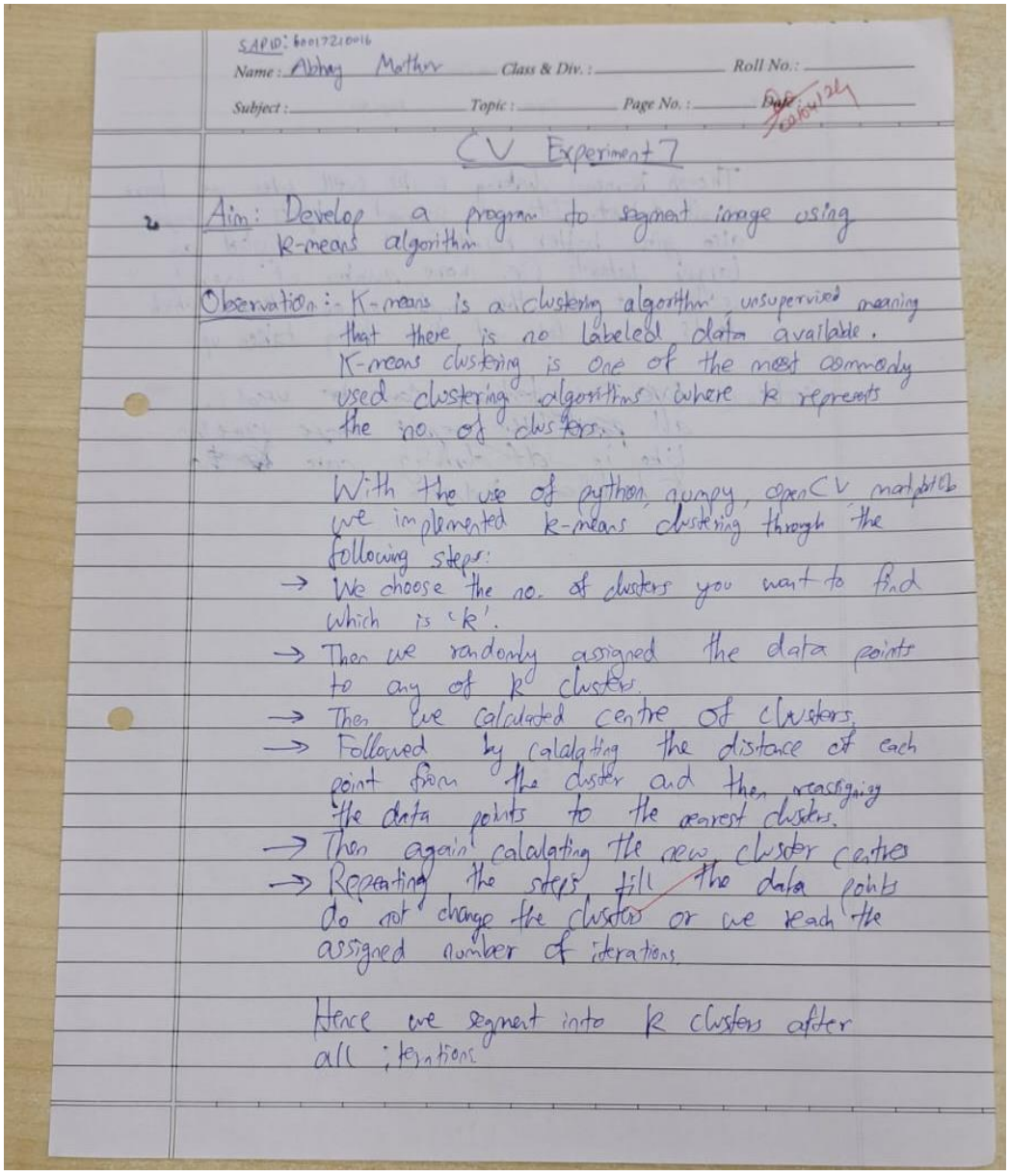
K Means is a clustering algorithm. Clustering algorithms are unsupervised algorithms which means that there is no labelled data available. It is used to identify different classes or clusters in the given data based on how similar the data is. Data points in the same group are more similar to other data points in that same group than those in other groups.

1. Choose the number of clusters you want to find which is k.
2. Randomly assign the data points to any of the k clusters.
3. Then calculate the centre of the clusters.
4. Calculate the distance of the data points from the centres of each of the clusters.
5. Depending on the distance of each data point from the cluster, reassign the data points to the nearest clusters.
6. Again calculate the new cluster centre.
7. Repeat steps 4,5 and 6 till data points don’t change the clusters, or till we reach the assigned number of iterations.

Observations

Department of Artificial Intelligence & Machine Learning

Academic Year 2023-2024



SAP ID: 60017210016
Name: Abhay Mathur Class & Div.: Roll No.:
Subject: Topic: Page No.: Date: 28/9/24

CV Experiment 7

Aim: Develop a program to segment image using k-means algorithm

Observation:- K-means is a clustering algorithm unsupervised meaning that there is no labeled data available. K-means clustering is one of the most commonly used clustering algorithms where k represents the no. of clusters.

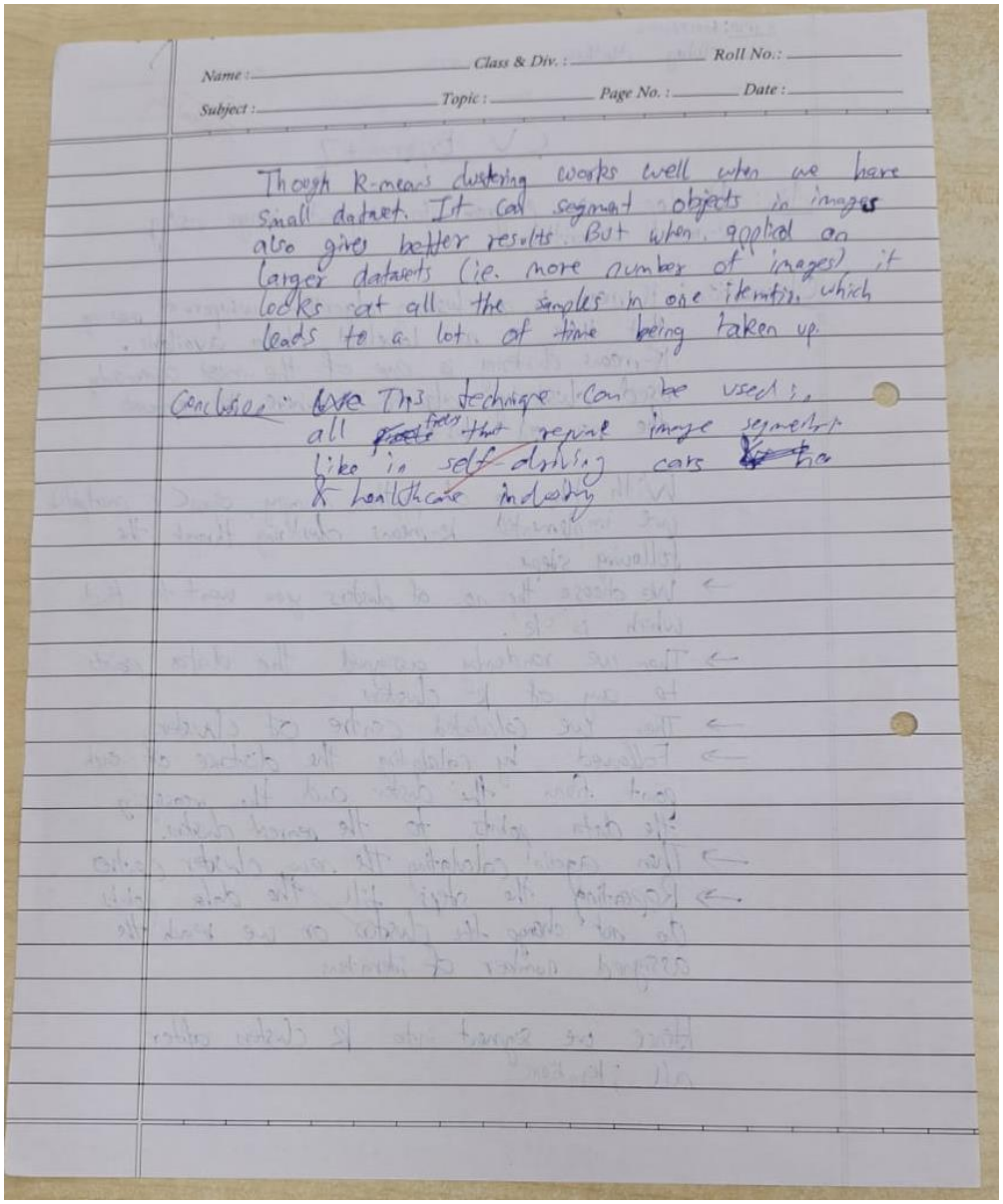
With the use of python, numpy, openCV matplotlib we implemented k-means clustering through the following steps:

→ We choose the no. of clusters you want to find which is 'k'.
→ Then we randomly assigned the data points to any of k clusters.
→ Then we calculated centre of clusters.
→ Followed by calculating the distance of each point from the cluster and then reassigning the data points to the nearest clusters.
→ Then again calculating the new cluster centres
→ Repeating the steps till the data points do not change the clusters or we reach the assigned number of iterations.

Hence we segment into k clusters after all iterations

Department of Artificial Intelligence & Machine Learning

Academic Year 2023-2024

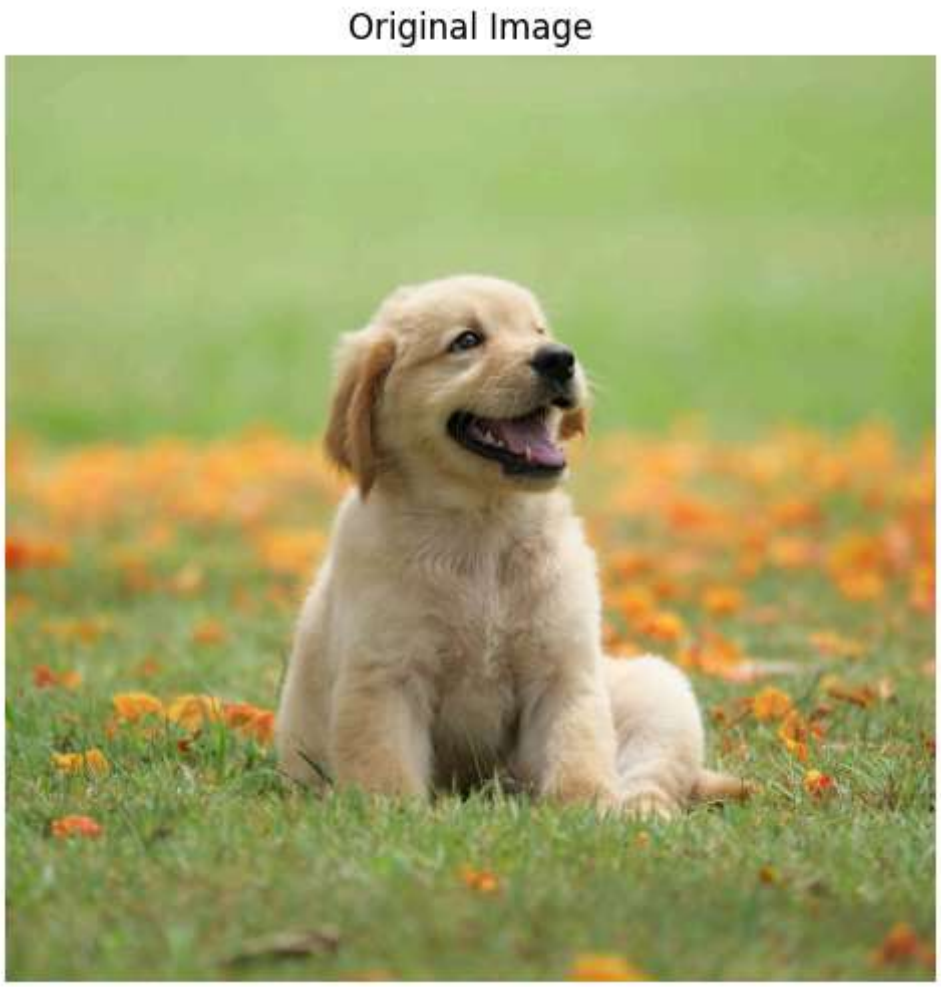


Though K-means clustering works well when we have small dataset. It can segment objects in images also gives better results But when applied on larger datasets (ie. more number of images) it looks at all the samples in one iteration which leads to a lot of time being taken up.

Conclusion: The technique can be used in all fields that require image segmentation like in self-driving cars & healthcare industry

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import cv2
```

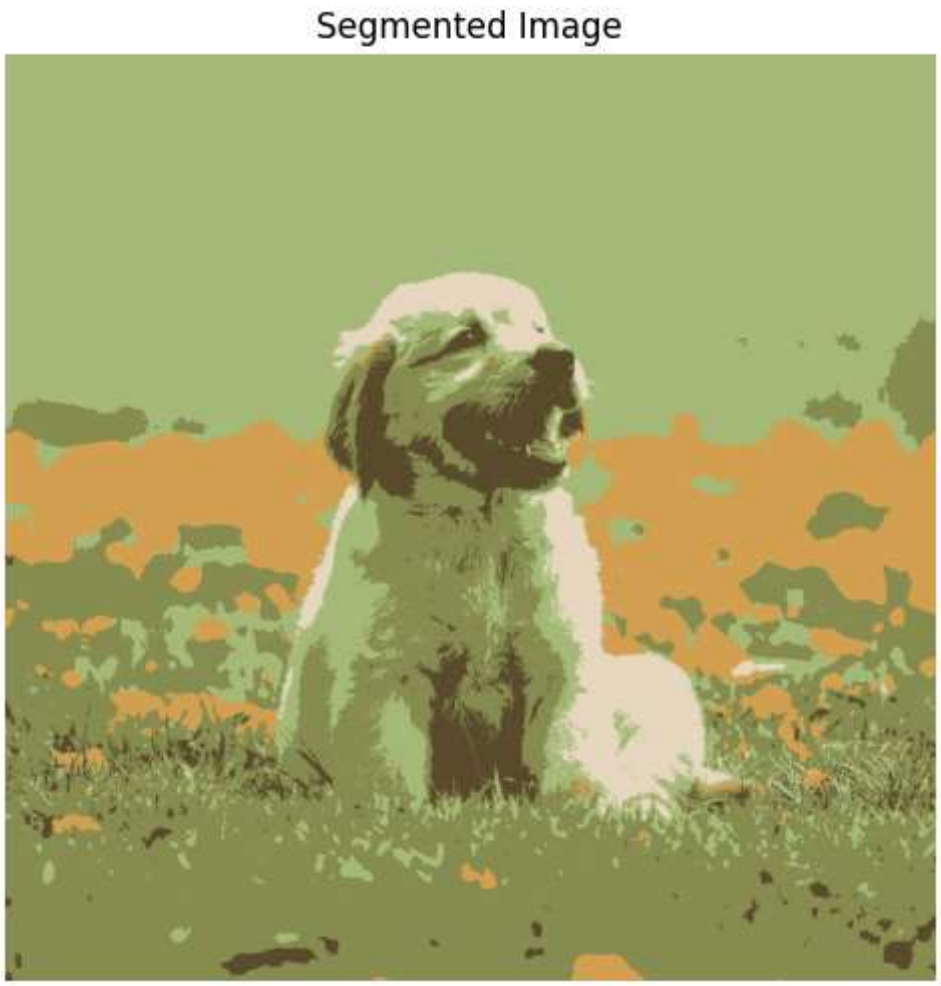
In [ ]:
```
# Load sample image
image_bgr = cv2.imread("test_image.jpg")
image = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(8, 6))
plt.imshow(image_rgb)
plt.title("Original Image")
plt.axis("off")
plt.show()
```



In [ ]:
```
# Reshape the image to a 2D array of pixels
w, h, d = original_shape = tuple(image.shape)
image_array = np.reshape(image, (w * h, d))

# Perform k-means clustering
n_colors = 5  # Number of clusters (colors)
kmeans = KMeans(n_clusters=n_colors, random_state=0)
kmeans.fit(image_array)
```

```
# Get labels and cluster centers
labels = kmeans.predict(image_array)
centers = kmeans.cluster_centers_

# Create segmented image using the cluster centers
segmented_image = np.zeros_like(image_array)
for i in range(len(labels)):
    segmented_image[i] = centers[labels[i]]

# Reshape segmented image to original shape
segmented_image = segmented_image.reshape((w, h, d))

# Display segmented image
plt.figure(figsize=(8, 6))
plt.imshow(segmented_image)
plt.title("Segmented Image")
plt.axis("off")
plt.show()
```

```
c:\Users\a21ma\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```



YOLO

You Only Look Once

By,

Shruti Shah
Vinayak Kundu
Abhay Mathur
Aashish Charaya
Devansh Rathor
Nidhi Tiwari

For, Computer Vision Group Assignment

What is YOLO?

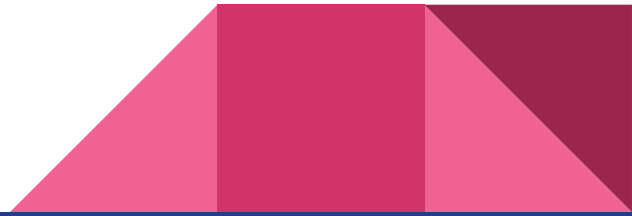
- YOLO, is a state-of-the-art, real-time object object detection and image segmentation model.
- Developed by Joseph Redmon and Ali Farhadi at the University of Washington.
- Launched in 2015, YOLO quickly gained popularity for its high speed and accuracy.

How does YOLO work?

- YOLO is an AI framework that supports multiple computer vision tasks.
- The framework can be used to perform detection, segmentation, obb, classification, and pose estimation.
- YOLO applies a single neural network to the full image.
- This network divides the image into regions and predicts bounding boxes and probabilities for each region.
- These bounding boxes are weighted by the predicted probabilities.
- These detections are thresholded to only see the highest scoring ones.

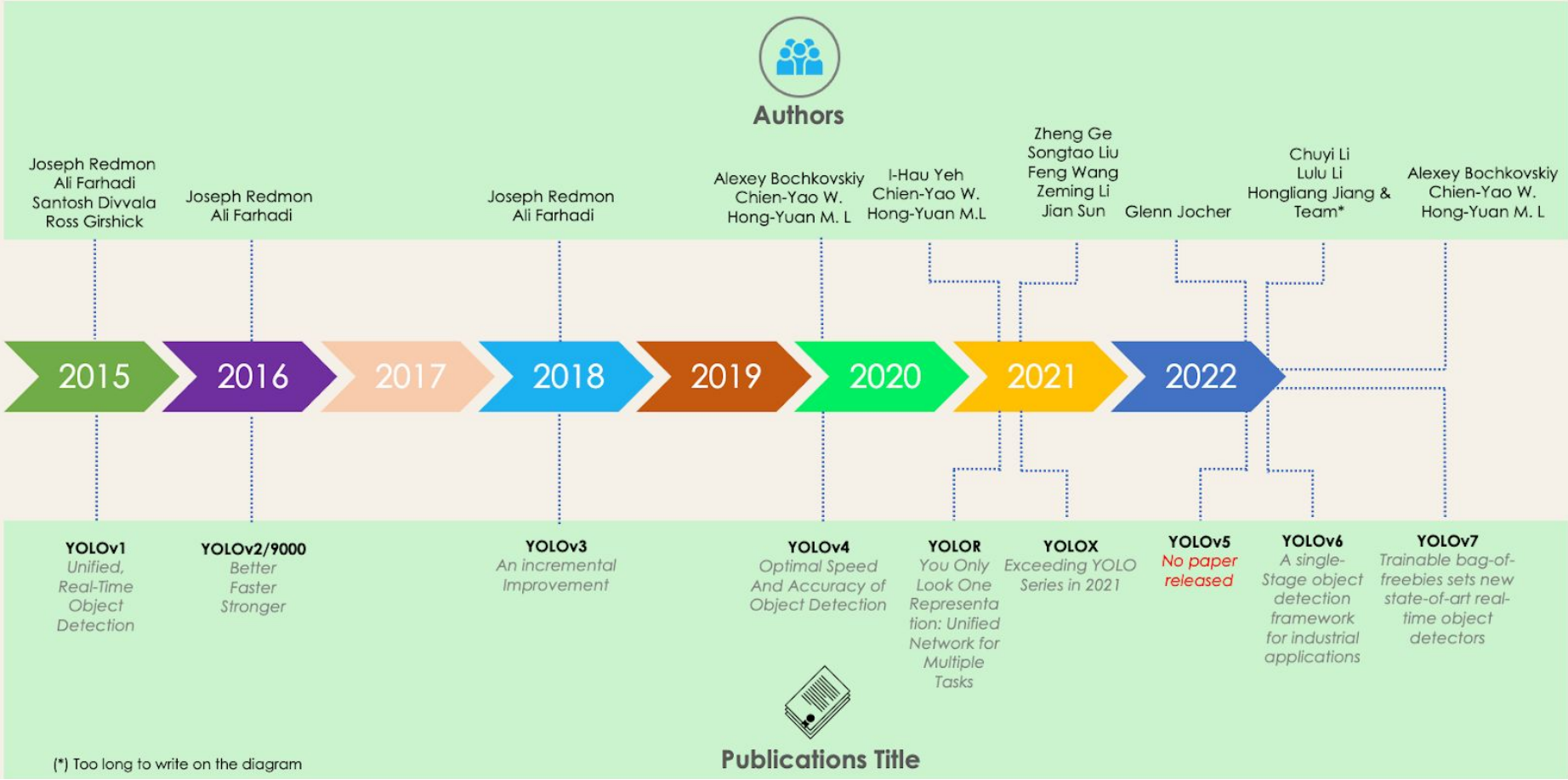
Where is YOLO used?

- **Security:** Monitors live feeds and identifies potential threats.
- **Healthcare:** Assists doctors by analyzing medical images in real-time.
- **Agriculture:** Scouts fields for crops, weeds, and diseases.
- **Self-Driving Cars:** Detects objects on the road like cars and pedestrians.
- **Robotics:** Enables robots to grasp objects and navigate their surroundings.



History of YOLO

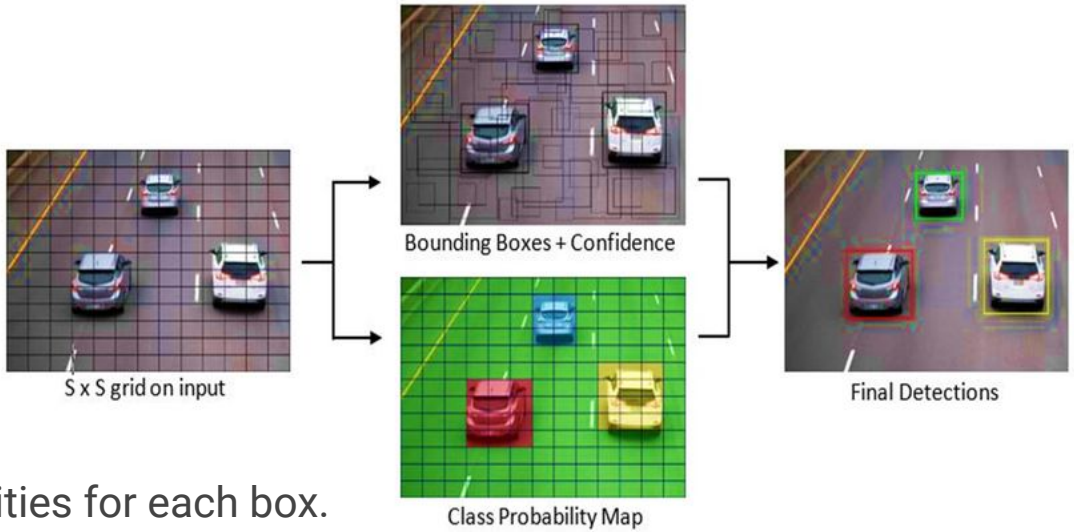
YOLO Timeline From 2015 to 2022



Methodology & Architecture

Methodology

- Grid-based Detection: Divide image into grid cells.
- Single Forward Pass: Detect objects in one go.
- Bounding Box Prediction: Predict bounding boxes for each cell.
- Class Prediction: Assign class probabilities for each box.
- Confidence Score: Rate likelihood of object presence.
- Non-max Suppression: Remove duplicate detections.

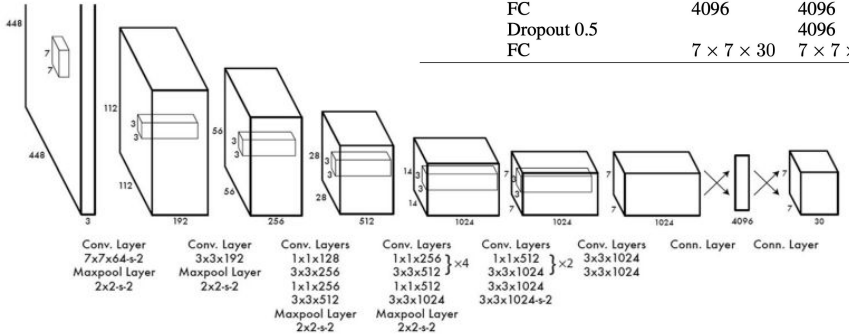


Architecture of YOLOv1

- First iteration of YOLO.
- Consisted of 24 convolution layers, 4 max-pooling layers and 2 fully connected layers at the end.
- Input size being 448*448 image.
- First Convolution layer of size 7 by 7
- Rest convolution layer are combinations of 1*1 and 3*3 size masks
- The Architecture uses Leaky-ReLu as activation function.
- Output for each class would be: (x,y,width,height,confidence)
- All the versions calculate 3 types of errors:
    - localization loss : Coordinates error
    - confidence loss : Probability calculation error
    - classification loss : Type of Class error

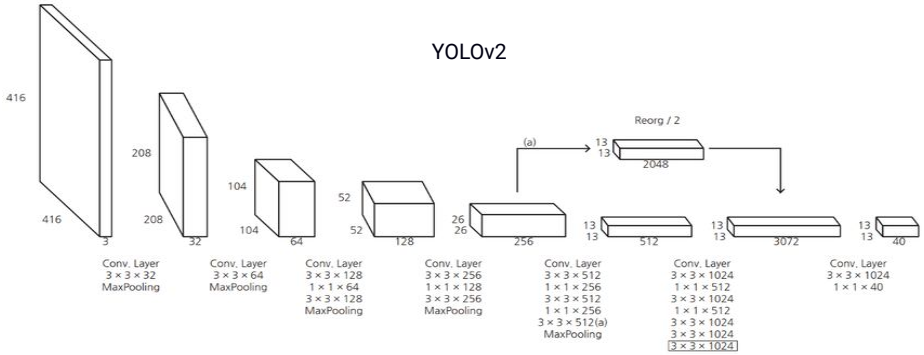
	Type	Filters	Size/Stride	Output
	Conv	64	$7 \times 7 / 2$	224×224
	Max Pool		$2 \times 2 / 2$	112×112
	Conv	192	$3 \times 3 / 1$	112×112
	Max Pool		$2 \times 2 / 2$	56×56
$1\times$	Conv	128	$1 \times 1 / 1$	56×56
	Conv	256	$3 \times 3 / 1$	56×56
	Conv	256	$1 \times 1 / 1$	56×56
	Conv	512	$3 \times 3 / 1$	56×56
	Max Pool		$2 \times 2 / 2$	28×28
$4\times$	Conv	256	$1 \times 1 / 1$	28×28
	Conv	512	$3 \times 3 / 1$	28×28
	Conv	512	$1 \times 1 / 1$	28×28
	Conv	1024	$3 \times 3 / 1$	28×28
	Max Pool		$2 \times 2 / 2$	14×14
$2\times$	Conv	512	$1 \times 1 / 1$	14×14
	Conv	1024	$3 \times 3 / 1$	14×14
	Conv	1024	$3 \times 3 / 1$	14×14
	Conv	1024	$3 \times 3 / 2$	7×7
	Conv	1024	$3 \times 3 / 1$	7×7
	Conv	1024	$3 \times 3 / 1$	7×7
	FC		4096	4096
	Dropout 0.5			4096
	FC		$7 \times 7 \times 30$	$7 \times 7 \times 30$

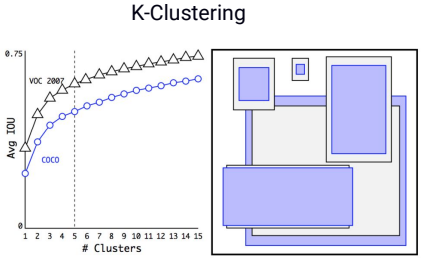
YOLOv1



Architecture of YOLOv2

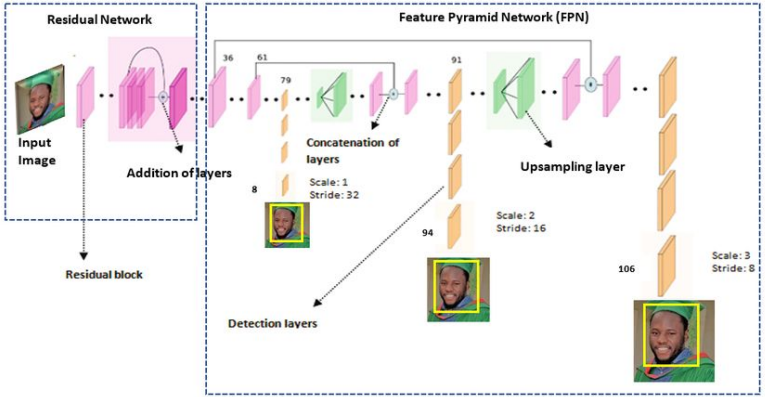
- Next iteration to YOLOv1, multiple changes made to the approach and architecture
- Input Image size is 416*416 instead of 448*448, to get 13*13 feature map
- Uses DarkNet19 that is 19 convolution layers instead of 24.
- 2 Fully connected layers replaced by convolution layers
- Anchoring boxes replaces bounding boxes by above action.
- Few features added like:
    - Batch Normalization
    - Breaking layers into smaller dimension for finer detail detection
    - K-clustering used for accurate box prediction





Architecture of YOLOv3

- Similar to YOLOv2 in terms of working, here are the changes.
- DarkNet 19 is replaced by DarkNet 53, A deeper convolution network with 53 layers
- Uses Feature Pyramid Network (FPN) to get features at multiple scale.

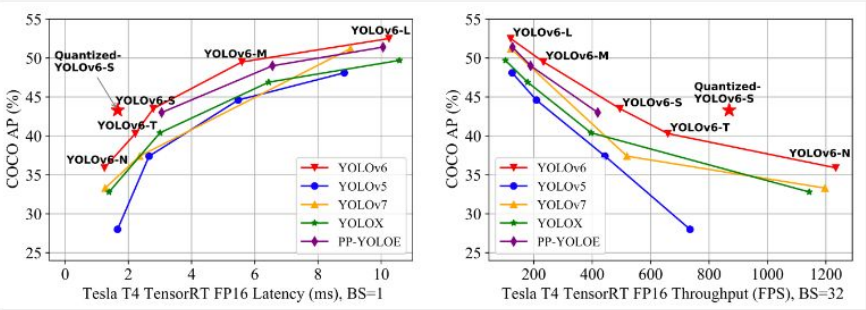


	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Table 1. **Darknet-53.**

Architecture changes in YOLOv4 to YOLOv7

- YOLOv4 replaced the Darknet 53 backbone with CSPDarkNet 53, which is 29 convolution layers with 27.6 million parameters Instead of incorporating FPN, it used PANnet for parameter aggregation for different level detection.
- YOLOv5, the one which wasn't documented like YOLOv1-4, first model based on pytorch, contained 4 sizes:
    - YOLOv5s (smallest)
    - YOLOv5m
    - YOLOv5l
    - YOLOv5x (largest).
- YOLOv6 and v7 were more improved versions, taking object detection to industrial levels



Demonstration

- Weapon Detection Dataset & Deployment
- Weapon Detection

Advantages & Limitations

Advantages

1. **Speed:** YOLO is known for its real-time processing capabilities, making it suitable for applications requiring rapid object detection.
2. **Simplicity:** Its single-stage architecture simplifies the process of object detection by directly predicting bounding boxes and class probabilities.
3. **Objectiveness:** YOLO considers the entire image at once, enabling it to detect objects in context and reducing false positives.
4. **Efficiency:** YOLO processes images in a single pass through the network, making it computationally efficient compared to other object detection methods.

Limitations

1. **Localization Accuracy:** YOLO may struggle with accurately localizing small or closely packed objects due to its coarse grid output.
2. **Lower Recall:** It may miss detecting small objects or objects with low contrast in cluttered scenes compared to two-stage detectors.
3. **Class Imbalance:** YOLO may face challenges in scenarios with significant class imbalance, affecting the accuracy of less frequent classes.
4. **Fixed Grid Size:** The fixed grid structure of YOLO can limit its ability to detect objects at varying scales effectively.

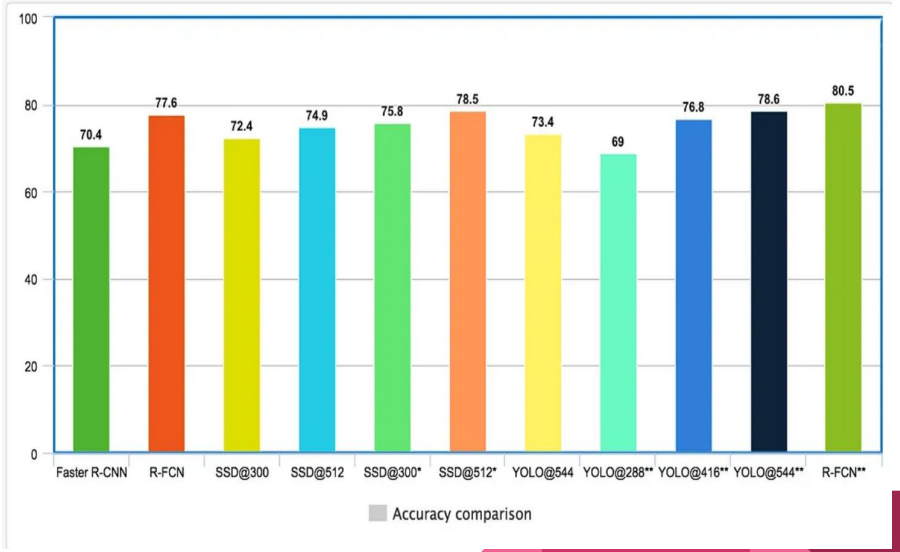
Comparative Analysis

Comparing YOLO with other models

FEATURES	**YOLO**	**FASTER R-CNN**	**SSD(Single Shot Multibox Detector)**	**RetinaNet**
TYPE	Single-stage detector	Two-stage detector	Single-stage detector	Single-stage detector
PROCESSING	Employs a single CNN to analyze the entire image at once	Two step process uses Regional Proposal Network(RPN) and Fast R-CNN detector	Analyzes the image by dividing it into grid of cells	Divides image into grid,predicts per cell using multi-scale features
KEY ELEMENTS	Single CNN predicts bounding boxes and class probabilities	RPN proposes regions and Faster R-CNN classifies objects and refines boxes	Uses multi-scale feature maps and default boxes	Improved rare object accuracy with focal loss and multi-scale detection with feature pyramid network(FPN)

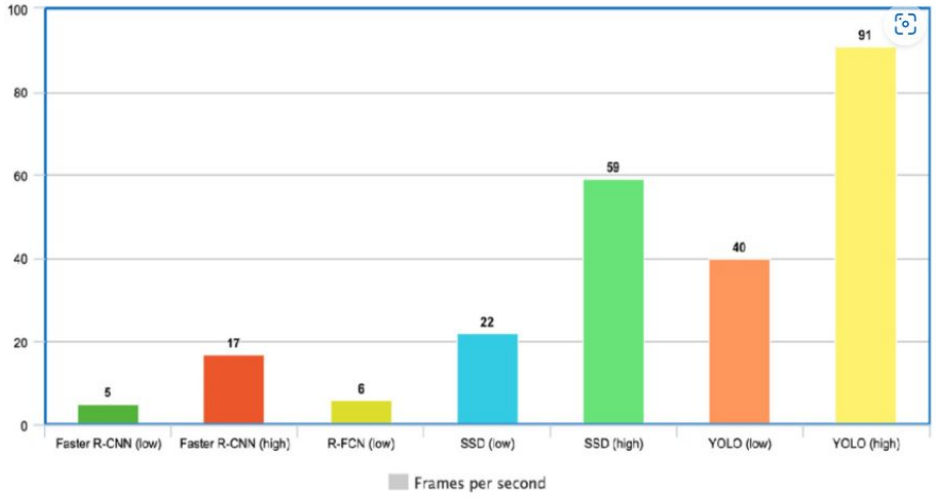
Accuracy comparison

- This graph compares the performance (mAP) of SSD, YOLO, Faster R-CNN and other object detection models on the PASCAL VOC 2012 dataset.
- For both SSD and YOLO, using higher resolution input images (e.g., 512x512 vs 300x300 for SSD) leads to better accuracy (higher mAP).
- At comparable image resolutions (e.g., 300x300), SSD generally achieves higher mAP compared to YOLO.



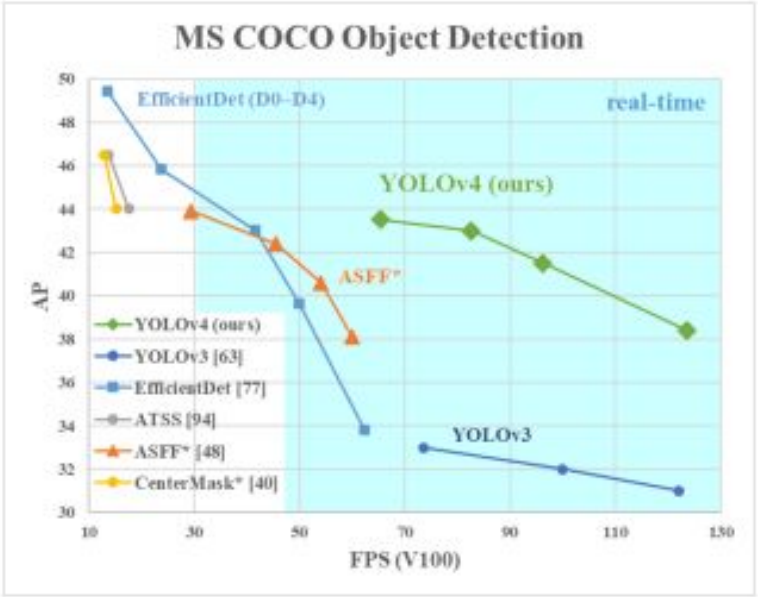
Real Time Object Detection

- The graph highlights a general trend where YOLO models are the fastest (highest FPS), followed by SSD, R-FCN, and Faster R-CNN.
- This is because YOLO utilizes a single-stage detection approach, making it computationally lighter compared to the two-stage approaches used by Faster R-CNN and R-FCN.
- Higher resolution input images lead to a decrease in FPS.



Speed vs. Accuracy Trade-Off in Real-Time Usage

- The graph represents comparison between different object detection models based on their mAP on the MS COCO dataset
- YOLO models (YOLOv4, YOLOv3, YOLO) achieve a good balance between speed and accuracy (mAP between 44% and 55%).
- EfficientDet models (EfficientDet-D7, EfficientDet-D0) offer higher mAP (around 59%) compared to YOLO models but may be slower.
- The real-time model (EfficientDer(DI-04)) achieves an mAP of 44%, similar to YOLO models. It likely offers faster performance than EfficientDet models but with lower accuracy.
- AVCVGG shows the highest mAP (around 76%) but may not be suitable for real-time applications due to its complex architecture. This model prioritizes accuracy over speed.



Questionnaire

Explain the methodology of YOLO

YOLO (You Only Look Once) is a popular object detection algorithm that is known for its speed and efficiency. Its methodology can be broken down into several key steps:

- **Image Division:** The input image is divided into a grid of cells. Each cell is responsible for predicting objects that fall within it.
- **Bounding Box Prediction:** Within each grid cell, YOLO predicts bounding boxes that enclose objects. Each bounding box is represented by a set of coordinates (x, y) for the center of the box, its width, and height (w, h). Additionally, YOLO predicts the confidence score for each bounding box, which indicates the likelihood that the box contains an object and the accuracy of the box's localization.
- **Object Classification:** YOLO simultaneously performs object classification for each bounding box. It assigns class probabilities to each box to determine the type of object it contains. This is typically done using a softmax function to output probabilities for each class.
- **Non-Maximum Suppression (NMS):** After obtaining multiple bounding boxes with confidence scores and class probabilities, YOLO applies non-maximum suppression to remove redundant and overlapping boxes. NMS ensures that only the most confident and accurate bounding boxes are retained for each object.
- **Loss Function:** During training, YOLO uses a combination of localization loss, confidence loss, and classification loss to optimize its parameters. The localization loss penalizes errors in bounding box coordinates, the confidence loss penalizes incorrect objectness predictions, and the classification loss penalizes errors in class predictions. The overall loss function is a weighted sum of these individual losses.

What are the advantages of yolo?

YOLO (You Only Look Once) has several advantages over other object detection approaches, which contribute to its popularity and widespread adoption:

- **Speed:** YOLO is highly efficient and capable of real-time object detection. By processing the entire image in a single pass through a convolutional neural network (CNN), YOLO eliminates the need for multiple passes and complex post-processing steps, resulting in faster inference speeds. This makes it suitable for applications where speed is crucial, such as video surveillance, autonomous driving, and robotics.

- **Global Context:** Unlike some other object detection methods that rely on sliding windows or region proposal techniques, YOLO considers the entire image during inference. By dividing the image into a grid and predicting objects within each grid cell, YOLO captures global context, enabling better understanding of object relationships and scene composition.

- **Unified Framework:** YOLO offers a unified framework for object detection and classification. Instead of treating object detection and classification as separate tasks, YOLO jointly predicts bounding boxes and class probabilities, which leads to more efficient use of computational resources and improved overall performance.

- **Flexibility:** YOLO is flexible and can be adapted to various tasks and scenarios. Its architecture can be modified and extended to handle different input sizes, aspect ratios, and numbers of classes. Additionally, YOLO can be optimized for deployment on different hardware platforms, including CPUs, GPUs, and specialized accelerators.

- **Accuracy-Performance Tradeoff:** YOLO achieves a good balance between accuracy and performance. While it may not always achieve state-of-the-art accuracy compared to some slower and more complex object detection methods, it offers competitive performance with significantly faster inference speeds, making it suitable for real-world applications where both speed and accuracy are important considerations.

Thank You

CV Mini Project

Name: Abhay Mathur
SAPID: 60017210016
Branch: AIML A1

Theory:

In this project, we have implemented a real-time hand gesture-controlled ping-pong game using the cvzone library for hand tracking and OpenCV for image processing. First, we capture video from our computer's camera and detect hands using a hand detector. Then, we resize the game window for better viewing. We import images for the game elements like the background, ball, bats, and game over screen. Within the game logic, we handle bat movement based on hand position, ball movement, collision detection, scoring, and game over conditions. The game loop continuously updates the frame with the game elements and displays the score. If we want to restart the game, we just need to press 'R', and to quit, we press 'Q'.

Code:

pong.py:

```
import numpy as np
import cv2 as cv
import cvzone
from cvzone.HandTrackingModule import HandDetector

# Captures the video feed from your computer's camera. 0 denotes the camera of
your laptop.
capture = cv.VideoCapture(0)
# Used to detect maximum two hands with confidence 0.8.
detector = HandDetector(detectionCon = 0.8, maxHands = 2)

# Usually, the game window pop-up will have 640 x 480 resolution. This is far too
small an area for our game.
# So, we have to resize our window.
# 3 denotes your width. It sets the width of the game window to 1280 pixels.
capture.set(3, 1280)
# 4 denotes your height. It sets the height of the game window to 720 pixels.
capture.set(4, 720)

# Importing all images. imread() helps to read an image from a path.
# cv.IMREAD_UNCHANGED is to ensure the image is read without any preprocessing
being done by the imread() function.
# This is needed for the transparency of the images to be maintained for the
"tools" of the game, so to say.
background = cv.imread(r'C:\Users\a21ma\OneDrive\Desktop\Code\Machine
Learning\Ping Pong Game\Resources\Background.png')
# background = cv.resize(background, 1280, 720)
ball = cv.imread(r'C:\Users\a21ma\OneDrive\Desktop\Code\Machine Learning\Ping Pong
Game\Resources\Ball.png',cv.IMREAD_UNCHANGED)
bat1 = cv.imread(r"C:\Users\a21ma\OneDrive\Desktop\Code\Machine Learning\Ping Pong
Game\Resources\bat1.png",cv.IMREAD_UNCHANGED)
bat2 = cv.imread(r"C:\Users\a21ma\OneDrive\Desktop\Code\Machine Learning\Ping Pong
Game\Resources\bat2.png",cv.IMREAD_UNCHANGED)
```

```
gameOver = cv.imread(r'C:\Users\a21ma\OneDrive\Desktop\Code\Machine Learning\Ping Pong Game\Resources\gameOver.png')

# print(background.shape)
# print(ball.shape)
# print(bat1.shape)
# print(bat2.shape)
# print(gameOver.shape)
# Set initial ball postion, score, a flag and the speeds
position = [100, 100]
isOver = False
speedX, speedY = 15, 15
score = [0, 0]

# To capture video footage from the camera.
while True:
    # read() function returns the frame and the boolean value that says whether the frame was read successfully or not.
    # We flip the image horizontally to avoid confusion due to lateral inversion.
    isTrue, frame = capture.read()
    frame = cv.flip(frame, 1)
    frame = cv.resize(frame,(1280,720))
    # To detect hands and its landmarks, without any annotations. findHands returns a boolean value.
    hands, frame = detector.findHands(frame, flipType = False)

    # For overlaying images.
    # addWeighted() puts the foreground image with an amount of transparency on top of the underlying image.
    # In this case, our underlying image at every instant is the camera feed.
    frame = cv.addWeighted(frame, 0.5, background, 0.5, 0.0)

    # Check for hands. Hands is a dictionary, so we process it like one.
    if hands:
        for hand in hands:
            # Gets the positional values of hand in bounding box.
            x, y, w, h = hand['bbox']
            h1, w1, _ = bat1.shape  # To get height and width of the bat
            y1 = y - h1//2  # To ensure your hands are at center of the bat.
            # Clips the value to ensure overlays don't occur outside the pop-up space.
            y1 = np.clip(y1, 20, 415)

            if hand['type'] == 'Left':
                left_bat_pos = 59
                frame = cvzone.overlayPNG(frame, bat1, (left_bat_pos, y1))

                if (left_bat_pos< position[0] < left_bat_pos + w1) and (y1 < position[1] < y1 + h1):
                    # If bat hits the ball, reverse the ball direction along x-axis.
                    speedX = -speedX
                    position[0] += 20
```

```
                    score[0] += 1

            if hand['type'] == 'Right':
                frame = cvzone.overlayPNG(frame, bat2, (1195, y1))
                if (1145 < position[0] < 1165) and (y1 < position[1] < y1 + h1):
                    speedX = -speedX
                    position[0] -= 20
                    score[1] += 1

    # If ball goes out of bounds, game's over.
    if position[0] < 40 or position[0] > 1195:
        isOver = True

    if isOver:
        frame = gameOver
        cv.putText(frame, str(score[1] + score[0]).zfill(2), (585, 360),
cv.FONT_HERSHEY_COMPLEX,
                   2.5, (200, 0, 200), 5)
    else:
        # Move the ball. If the ball hits the wall of upper rectangle, reverse its
direction along y-axis.
        if position[1] >= 500 or position[1] <= 10:
            speedY = -speedY

        # We give the ball speed.
        position[0] += speedX
        position[1] += speedY

        # print(frame.shape)
        # Draw the ball
        frame = cvzone.overlayPNG(frame, ball, position)
        # Display score as game goes on. we use putText() for this. (255, 255,
255) is white colour
        cv.putText(frame, str(score[0]), (300, 650),
                   cv.FONT_HERSHEY_COMPLEX, 3, (255, 255, 255), 5)
        cv.putText(frame, str(score[1]), (900, 650),
                   cv.FONT_HERSHEY_COMPLEX, 3, (255, 255, 255), 5)

    # imshow() function returns the matrix of pixels in a new window. It also
takes the name of the popup window.
    cv.imshow('Pong game', frame)
    # waitKey() function allows the popup window to be displayed for a certain
time period in milliseconds.
    key = cv.waitKey(1)
    # If R is pressed, re-initialise the respective variables and restart the
game.
    if key == ord('r'):
        position = [100, 100]
        speedX = 15
        speedY = 15
        isOver = False
        score = [0, 0]
```

```
        gameOver = cv.imread(r'C:\Users\a21ma\OneDrive\Desktop\Code\Machine
Learning\Ping Pong Game\Resources\gameOver.png')

    # To quit the game
    if key == ord('q'):
        break

# Release the capture and close all OpenCV windows
capture.release()
cv.destroyAllWindows()
```

Output: