RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI NAGAR, THANDALAM - 602 105

AI23A37 – COMPUTER VISION AND ITS APPLICATIONS

LABORATORY LAB MANUAL

NAME: HARISH KUMAR V

REGISTER NUMBER: 2116-231501057

YEAR / BRANCH / SECTION: III YEAR / AIML / A

SEMESTER: V SEMESTER

ACADEMIC YEAR: 2025-2026

BONAFIDE CERTIFICATE

CERTIFIED THAT THIS LABORATORY RECORD REPORT FOR "**COMPUTER VISION & ITS APPLICATIONS**" IS THE BONAFIDE WORK OF "**HARISH KUMAR V** [231501057]" WHO CARRIED OUT THE PRACTICAL WORK UNDER MY SUPERVISION.

Submitted for the Practical Examination held on 05/11/2025

SIGNATURE

Mrs. Sangeetha K, AIML,

REC (Autonomous) Thandalam,

Chennai - 602 105

INTERNAL EXAMINER **EXTERNAL EXAMINER**

TABLE OF CONTENTS

REG NO: 231501057 **NAME:** HARISHKUMAR V

YEAR: III YEAR **BRANCH**: AIML **SEC**: A

EXPT NO: 1	BASIC IMAGE PROCESSING OPERATIONS
DATE: 08/07/2025	

AIM:

To Implement various basic image processing operations like Reading image, writing image and conversion of images.

ALGORITHM:

1. Import required libraries **(OpenCV, NumPy).**
2. Read the input image using **cv2.imread().**
3. Display the image using **cv2.imshow().**
4. Convert colour spaces (RGB ↔ Grayscale ↔ HSV) using **cv2.cvtColor().**
5. Write the output image using **cv2.imwrite().**
6. Close windows using **cv2.destroyAllWindows().**

CODE:

```
#PROGRAM 1


import cv2
from google.colab.patches import cv2_imshow

# ---- 1. Reading an Image ----
# Replace 'example.jpg' with your actual image path
image = cv2.imread('/content/rec.jpg')

# Check if the image was successfully loaded
if image is None:
    print("Error: Image not found.")
    exit()

# Display the original image
cv2_imshow(image)
```

```
# cv2.waitKey(0) # waitKey and destroyAllWindows are not needed with
cv2_imshow
# cv2.destroyAllWindows()


# ---- 2. Writing (Saving) the Image ----
cv2.imwrite('/content/rec.jpg', image)
print("Image saved as output.jpg")

# ---- 3a. Convert to Grayscale ----
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2_imshow(gray_image)
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# ---- 3b. Convert to HSV ----
hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
cv2_imshow(hsv_image)
# cv2.waitKey(0)
# cv2.destroyAllWindows()


# ---- 3c. Convert to Binary (Thresholding) ----
# Threshold value = 127, Max value = 255
_, binary_image = cv2.threshold(gray_image, 127, 255, cv2.THRESH_BINARY)
cv2_imshow(binary_image)
# cv2.waitKey(0)
# cv2.destroyAllWindows()
```

OUTPUT:

RESULT:

Thus, the various basic image processing operations like Reading image, writing image and conversion of images were implemented successfully.

EXPT NO: 2	CONTRAST ADJUSTMENT
DATE: 11-07-2025	

AIM:

To Implement contrast adjustment of an image.

ALGORITHM:

1. Read the input image in **grayscale.**
2. Normalize pixel values or use **histogram stretching**.
3. Apply **formula**: output = alpha * input + beta (alpha = contrast, beta = brightness).
4. Clip values to [0,255] range.
5. Display **original** and **adjusted images.**
6. Save the **adjusted image**.

CODE:

```
# PROGRAM 2

import cv2
import numpy as np
from google.colab.patches import cv2_imshow # Import cv2_imshow

# Step 1: Read the input image
image = cv2.imread('/content/PEACOCK.jpg')  # Replace with your image path

# Check if the image was successfully loaded
if image is None:
    print("Error: Image not found.")
    exit()

# Set the contrast factor (alpha)
alpha_increase = 1.5  # Contrast > 1: Increase contrast
alpha_decrease = 0.5  # 0 < Contrast < 1: Decrease contrast
beta = 0              # No brightness adjustment
```

```
# Step 2: Adjust the contrast
# To increase contrast
contrasted_img_inc = cv2.convertScaleAbs(image, alpha=alpha_increase,
beta=beta)
# To decrease contrast
contrasted_img_dec = cv2.convertScaleAbs(image, alpha=alpha_decrease,
beta=beta)

# Step 3: Display the images
cv2_imshow(image) # Use cv2_imshow
cv2_imshow(contrasted_img_inc) # Use cv2_imshow
cv2_imshow(contrasted_img_dec) # Use cv2_imshow
# cv2.waitKey(0) # waitKey and destroyAllWindows are not needed with
cv2_imshow
# cv2.destroyAllWindows() # waitKey and destroyAllWindows are not needed with
cv2_imshow

# Step 4: Save the results
cv2.imwrite('contrast_increased.jpg', contrasted_img_inc)
cv2.imwrite('contrast_decreased.jpg',  contrasted_img_dec)
print("Images saved as contrast_increased.jpg and contrast_decreased.jpg")
```

OUTPUT:

RESULT:

Thus, contrast adjustment of an image implemented successfully.

EXPT NO: 3	HISTOGRAM PROCESSING AND EQUALIZATION
DATE: 18-07-2025	

AIM:

To Implement Histogram processing and Equalization.

ALGORITHM:

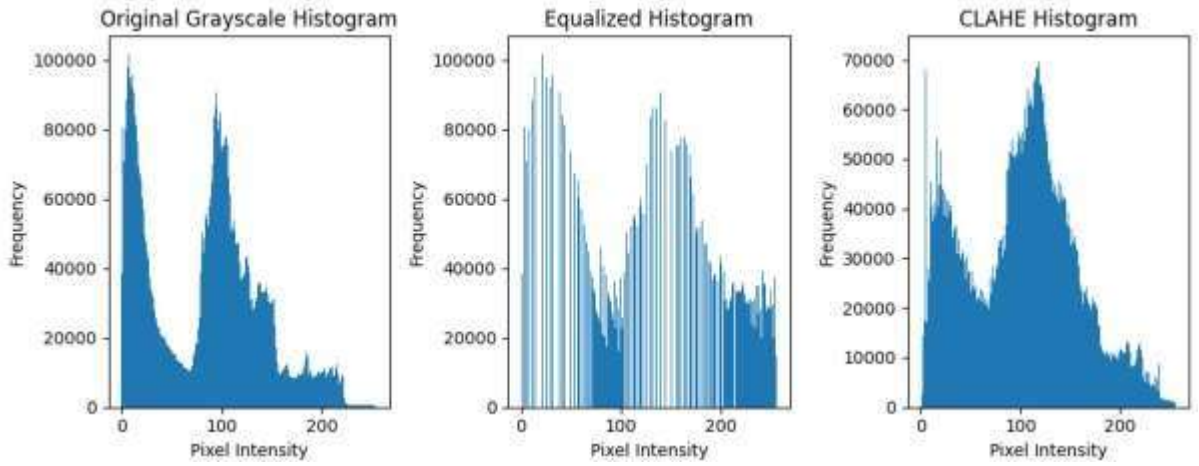
1. Convert image to grayscale.
2. Compute histogram using **cv2.calcHist().**
3. Normalize and plot the histogram.
4. Apply histogram equalization using **cv2.equalizeHist().**
5. Compare before and after histograms.
6. Display **results.**

CODE:

```
# PROGRAM 3
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow # Import cv2_imshow for Colab
compatibility

# Step 2: Read the input image
image = cv2.imread('/content/EMPIRE STATE BUILDING.avif')  # Replace with your
image path

# Check if image is loaded properly
if image is None:
    print("Error: Image not found.")
    exit()

# Step 3: Convert to grayscale
```

```
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Step 4: Compute histogram of original grayscale image
plt.figure(figsize=(10,4))
plt.subplot(1,3,1)
plt.title('Original Grayscale Histogram')
plt.hist(gray_image.ravel(), 256, [0,256])
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')

# Step 5: Apply global histogram equalization
equalized_image = cv2.equalizeHist(gray_image)

# Histogram for Equalized Image
plt.subplot(1,3,2)
plt.title('Equalized Histogram')
plt.hist(equalized_image.ravel(), 256, [0,256])
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')

# Step 6: Apply CLAHE (Contrast Limited Adaptive Histogram Equalization)
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
clahe_image = clahe.apply(gray_image)

# Histogram for CLAHE Image
plt.subplot(1,3,3)
plt.title('CLAHE Histogram')
plt.hist(clahe_image.ravel(), 256, [0,256])
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()

# Step 7: Display original, equalized and CLAHE images
cv2_imshow(gray_image) # Use cv2_imshow
cv2_imshow(equalized_image) # Use cv2_imshow
cv2_imshow(clahe_image) # Use cv2_imshow


# Step 8: Wait for key press and close windows
# cv2.waitKey(0) # waitKey and destroyAllWindows are not needed with
cv2_imshow
# cv2.destroyAllWindows() # waitKey and destroyAllWindows are not needed with
cv2_imshow
```

OUTPUT:


Original Grayscale Histogram
Equalized Histogram
CLAHE Histogram
Frequency
Pixel Intensity
0
20000
40000
60000
80000
100000
10000
30000
50000
70000
100
200

RESULT:

Thus, Histogram processing and Equalization implemented successfully.

EXPT NO: 4	LOW PASS AND HIGH PASS FILTERING MECHANISMS
DATE: 25-07-2025	

AIM:

To Implement the various low pass and high pass filtering mechanisms.

ALGORITHM:

1. Read and convert image to **grayscale.**
2. Apply low pass (Gaussian/average) filter using cv2.GaussianBlur() or **cv2.blur().**
3. Apply high pass filter using Laplacian or Sobel operator.
4. Combine results to observe **smoothing vs sharpening effects.**
5. Display **filtered images.**
6. Save **outputs.**

CODE:

```
# PROGRAM 4


import cv2
import numpy as np

# Step 2: Read the input image
image = cv2.imread('/content/SUNSET.jpg')  # Replace with your image path
from google.colab.patches import cv2_imshow

# Step 3: Convert to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Step 4: Apply Low Pass Filters
# a. Average Filter
avg_blur = cv2.blur(gray_image, (5, 5))

# b. Gaussian Filter
```

```
gaussian_blur = cv2.GaussianBlur(gray_image, (5, 5), 0)

# c. Median Filter
median_blur = cv2.medianBlur(gray_image, 5)

# Step 5: Apply High Pass Filters
# a. Laplacian Filter
laplacian = cv2.Laplacian(gray_image, cv2.CV_64F)
laplacian_abs = cv2.convertScaleAbs(laplacian)

# b. Sobel Filter (Horizontal and Vertical)
sobelx = cv2.Sobel(gray_image, cv2.CV_64F, 1, 0, ksize=5)  # Horizontal edges
sobely = cv2.Sobel(gray_image, cv2.CV_64F, 0, 1, ksize=5)  # Vertical edges
sobelx_abs = cv2.convertScaleAbs(sobelx)
sobely_abs = cv2.convertScaleAbs(sobely)

# Step 6: Display all filtered images
cv2_imshow(gray_image)
cv2_imshow(avg_blur)
cv2_imshow(gaussian_blur)
cv2_imshow(median_blur)
cv2_imshow(laplacian_abs)
cv2_imshow(sobelx_abs)
cv2_imshow(sobely_abs)


# Step 7: Wait for key press and close all windows
# cv2.waitKey(0)
# cv2.destroyAllWindows()
```

OUTPUT:

RESULT:

Thus, the various low pass and high pass filtering mechanisms was successfully implemented.

EXPT NO: 5	FOURIER TRANSFORM FOR FILTERING THE IMAGE
DATE: 01/08/2025	

AIM:

To implement the Use of Fourier transform for filtering the image.

ALGORITHM:

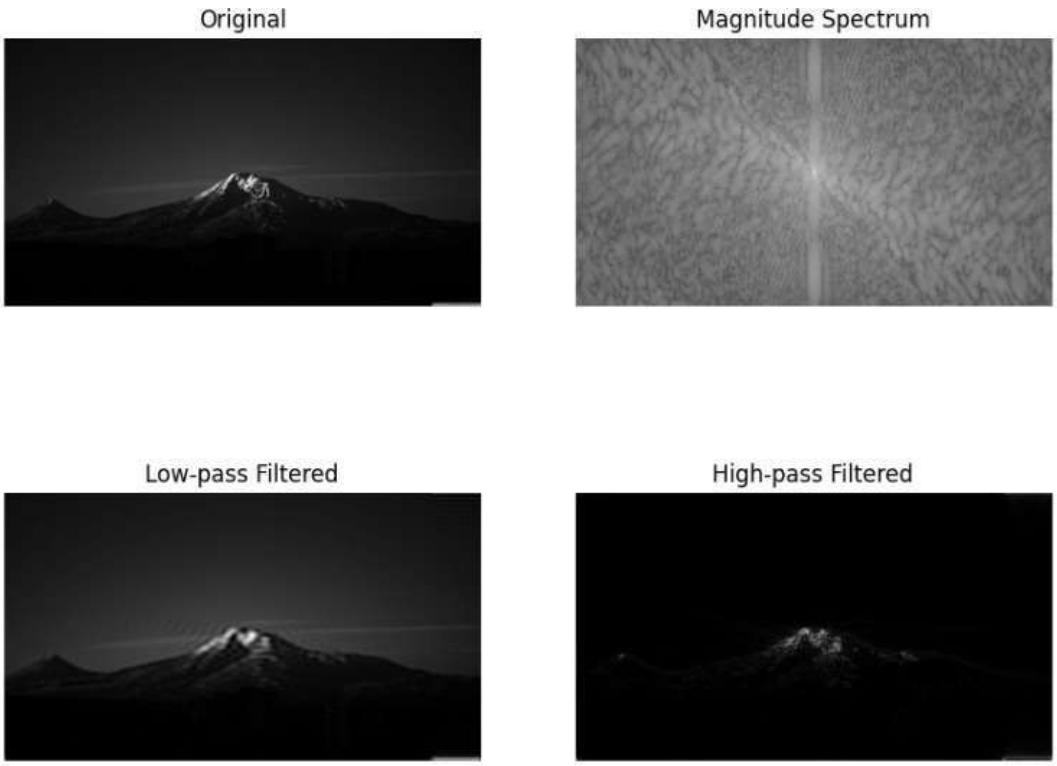
1. Convert image to **grayscale** and **float type.**
2. Apply DFT using **np.fft.fft2()** and shift using **np.fft.fftshift().**
3. Create a mask for low/high pass filtering in frequency domain.
4. Apply mask and perform inverse FFT using **np.fft.ifft2().**
5. Take **magnitude** and **display filtered image**.
6. Compare with **original.**

CODE:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read image
img = cv2.imread('/content/drive/MyDrive/input.jpg', 0)

# Check if the image was loaded successfully
if img is None:
    print("Error: Could not load image. Please check the file path and ensure
the image exists.")
else:
    # Fourier Transform
    f = np.fft.fft2(img)
    fshift = np.fft.fftshift(f)
    magnitude_spectrum = 20*np.log(np.abs(fshift))
```

```
# --- Low-pass filter ---
rows, cols = img.shape
crow, ccol = rows//2, cols//2
mask_lp = np.zeros((rows, cols), np.uint8)
r = 50  # radius for LPF
cv2.circle(mask_lp, (ccol, crow), r, 1, -1)

fshift_lp = fshift * mask_lp
img_lp = np.fft.ifft2(np.fft.ifftshift(fshift_lp))
img_lp = np.abs(img_lp)

# --- High-pass filter ---
mask_hp = 1 - mask_lp
fshift_hp = fshift * mask_hp
img_hp = np.fft.ifft2(np.fft.ifftshift(fshift_hp))
img_hp = np.abs(img_hp)

# Display results
plt.figure(figsize=(10,8))
plt.subplot(2,2,1), plt.imshow(img, cmap='gray')
plt.title("Original"), plt.axis('off')
plt.subplot(2,2,2), plt.imshow(magnitude_spectrum, cmap='gray')
plt.title("Magnitude Spectrum"), plt.axis('off')
plt.subplot(2,2,3), plt.imshow(img_lp, cmap='gray')
plt.title("Low-pass Filtered"), plt.axis('off')
plt.subplot(2,2,4), plt.imshow(img_hp, cmap='gray')
plt.title("High-pass Filtered"), plt.axis('off')
plt.show()
```

OUTPUT:



RESULT:

Thus, Fourier transform for filtering the image was implemented successfully.

EXPT NO: 6	SIFT AND HOG FEATURES FOR IMAGE ANALYSIS
DATE: 22/08/2025	

AIM:

To perform the Utilization of SIFT and HOG features for image analysis.

ALGORITHM:

1. Read image and convert to grayscale.
2. Initialize **SIFT/HOG detector** (cv2.SIFT_create(), cv2.HOGDescriptor()).
3. Detect keypoints and **compute descriptors**.
4. Draw **keypoints** on the image.
5. Display or store the **extracted features**.
6. Use features for **comparison or matching.**

CODE:

```
import cv2
import matplotlib.pyplot as plt
from skimage.feature import hog
from skimage import color

# ----------- SIFT -------------------
img = cv2.imread('/content/drive/MyDrive/input.jpg')

# Check if image was loaded successfully
if img is None:
    print("Error: Could not load image from the specified path. Please check
the file path and ensure the image exists.")
else:
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # SIFT detector
```

```
    sift = cv2.SIFT_create()
    keypoints, descriptors = sift.detectAndCompute(gray, None)

    # Draw keypoints
    sift_img = cv2.drawKeypoints(img, keypoints, None,
flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

    # ------------------ HOG -----------
    # Convert to grayscale for HOG
    gray_hog = color.rgb2gray(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))

    # Compute HOG features and visualization
    hog_features, hog_img = hog(gray_hog, orientations=9, pixels_per_cell=(8,
8),
                                cells_per_block=(2, 2), visualize=True,
channel_axis=None)

    # ------------------ Display Results -----------
    plt.figure(figsize=(10,5))

    plt.subplot(1,2,1)
    plt.imshow(cv2.cvtColor(sift_img, cv2.COLOR_BGR2RGB))
    plt.title("SIFT Features")
    plt.axis('off')

    plt.subplot(1,2,2)
    plt.imshow(hog_img, cmap='gray')
    plt.title("HOG Features")
    plt.axis('off')

    plt.show()
```

OUTPUT:

RESULT:

Thus, Utilization of SIFT and HOG features for image analysis was implemented successfully.

EXPT NO: 7	VARIOUS IMAGE SEGMENTATION ALGORITHM
DATE: 12/09/2025	

AIM:

To Implement the various image segmentation algorithm

ALGORITHM:

1. Read and **preprocess image** (resize, blur).
2. Apply **thresholding** (Otsu/manual) or clustering (K-means).
3. Label segments using connected component analysis.
4. Optionally use edge detection before segmentation.
5. Visualize segmented regions with colors.
6. Display **final segmented image**.

CODE:

```
import cv2, matplotlib.pyplot as plt

img = cv2.imread('image.jpg', 0)
edges = cv2.Canny(cv2.GaussianBlur(img, (5,5), 0), 100, 200)
plt.imshow(edges, cmap='gray'); plt.title("Canny Edges"); plt.show()
import cv2, numpy as np, matplotlib.pyplot as plt
img = cv2.imread('image.jpg', 0); h, w = img.shape
seed, th, mask = (h//2, w//2), 5, np.zeros_like(img)
stack = [seed]
while stack:
    x, y = stack.pop()
    if 0<=x<h and 0<=y<w and mask[x,y]==0 and abs(int(img[x,y])-
int(img[seed]))<th:
        mask[x,y]=255; stack += [(x+dx,y+dy) for dx in [-1,0,1] for dy in [-
1,0,1]]
plt.imshow(mask, cmap='gray'); plt.title("Region Growing"); plt.show()
```

```
import cv2, numpy as np, matplotlib.pyplot as plt
img = cv2.imread('image.jpg'); gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
_, th = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
dist = cv2.distanceTransform(th,1,5); _, fg =
cv2.threshold(dist,0.7*dist.max(),255,0)
bg = cv2.dilate(th,(3,3),3); unk =
cv2.subtract(bg.astype(np.uint8),fg.astype(np.uint8)); _, markers =
cv2.connectedComponents(np.uint8(fg))
markers = cv2.watershed(img, markers+1); img[markers==-1]=(255,0,0)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)); plt.title("Watershed");
plt.show()

from skimage import io, segmentation, color
img = io.imread('image.jpg')
seg = segmentation.felzenszwalb(img, scale=100, sigma=0.5, min_size=50)
io.imshow(color.label2rgb(seg, img)); io.show()

import cv2, numpy as np, matplotlib.pyplot as plt
img = cv2.imread('image.jpg'); mask = np.zeros(img.shape[:2], np.uint8)
bgd, fgd = np.zeros((1,65), np.float64), np.zeros((1,65), np.float64)
rect = (50,50,img.shape[1]-100,img.shape[0]-100)
cv2.grabCut(img, mask, rect, bgd, fgd, 5, cv2.GC_INIT_WITH_RECT)
mask2 = np.where((mask==2)|(mask==0),0,1).astype('uint8'); res =
img*mask2[:,:,None]
plt.imshow(cv2.cvtColor(res, cv2.COLOR_BGR2RGB)); plt.title("GrabCut");
plt.show()
```

OUTPUT:

Region Growing

Watershed

Felzenszwalb

GrabCut

RESULT:

Thus, various image segmentation algorithm was implemented successfully.

EXPT NO: 8	OPTICAL FLOW COMPUTATION ALGORITHM
DATE: 19/09/2025	

.

AIM:

Implement optical flow computation algorithm.

ALGORITHM:

1. Read two **consecutive video frames.**
2. Convert both to **grayscale.**
3. Compute optical flow using **cv2.calcOpticalFlowFarneback().**
4. Visualize motion vectors using **color coding.**
5. Overlay flow on **original image**.
6. Display **motion visualization.**

CODE:

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow

cap = cv2.VideoCapture('video.mp4')
ret, old = cap.read()
old_gray = cv2.cvtColor(old, cv2.COLOR_BGR2GRAY)

# Initial feature points
p0 = cv2.goodFeaturesToTrack(old_gray, maxCorners=50, qualityLevel=0.3,
minDistance=7)
mask = np.zeros_like(old)

# Parameters for Lucas-Kanade optical flow
lk = dict(winSize=(15, 15), maxLevel=2,
          criteria=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10,
0.03))
```

```
frame_count = 0
while True:
    ret, frame = cap.read()
    if not ret:
        break
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Calculate optical flow
    p1, st, _ = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None, **lk)

    # Select good points
    good_new = p1[st == 1]
    good_old = p0[st == 1]

    # Draw motion vectors every 5th frame only
    if frame_count % 5 == 0:
        mask[:] = 0  # clear mask for clean output
        for (new, old_pt) in zip(good_new, good_old):
            a, b = new.ravel()
            c, d = old_pt.ravel()
            mask = cv2.line(mask, (int(a), int(b)), (int(c), int(d)), (0, 255,
0), 2)
            frame = cv2.circle(frame, (int(a), int(b)), 3, (0, 0, 255), -1)
        motion = np.mean(np.linalg.norm(good_new - good_old, axis=1))
        print(f"Frame {frame_count}: Avg motion = {motion:.2f}")
        cv2_imshow(cv2.add(frame, mask))

    # Update previous frame and points
    old_gray = frame_gray.copy()
    p0 = good_new.reshape(-1, 1, 2)
    frame_count += 1

    if cv2.waitKey(30) & 0xFF == 27:  # ESC to quit
        break

cap.release()
cv2.destroyAllWindows()
```

OUTPUT:

RESULT:

Thus, optical flow computation algorithm was implemented successfully.

EXPT NO: 9	FACE DETECTION
EXPT NO: 26/09/2025	

AIM:

To implement the Face Detection on available online human face image datasets

ALGORITHM:

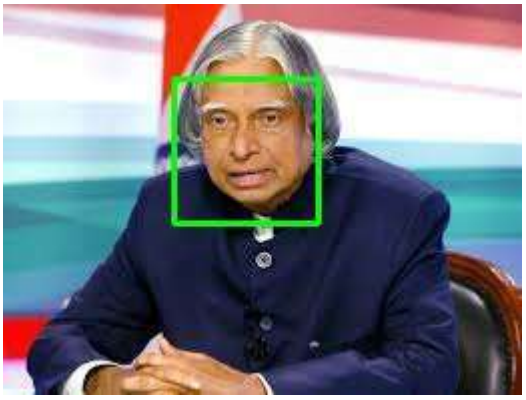
1. Load pretrained Haar Cascade or DNN face detector.
2. Read image or video frame.
3. Convert to grayscale for faster detection.
4. Detect faces using detectMultiScale().
5. Draw bounding boxes around faces.
6. Display and save output.

CODE:

```
import cv2

from google.colab.patches import cv2_imshow
face_cascade =
cv2.CascadeClassifier(cv2.data.haarcascades+'haarcascade_frontalface_default.x
ml')
img = cv2.imread('face.jpg'); gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
for (x,y,w,h) in faces: cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
cv2_imshow(img)

import dlib, cv2
from google.colab.patches import cv2_imshow
detector = dlib.get_frontal_face_detector()
img = cv2.imread('face.jpg'); rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
for d in detector(rgb):
cv2.rectangle(img,(d.left(),d.top()),(d.right(),d.bottom()),(0,255,0),2)
cv2_imshow(img); cv2.waitKey(0); cv2.destroyAllWindows()
```

```
from mtcnn import MTCNN
import cv2
from google.colab.patches import cv2_imshow
detector = MTCNN(); img = cv2.imread('face.jpg')
res = detector.detect_faces(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
for r in res: x,y,w,h = r['box'];
cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
cv2_imshow(img); cv2.waitKey(0); cv2.destroyAllWindows()

import cv2
from google.colab.patches import cv2_imshow

net =
cv2.dnn.readNetFromCaffe('deploy.prototxt','res10_300x300_ssd_iter_140000.caff
emodel')
img = cv2.imread('face.jpg'); h,w = img.shape[:2]
blob =
cv2.dnn.blobFromImage(cv2.resize(img,(300,300)),1.0,(300,300),(104,177,123))
net.setInput(blob); dets = net.forward()
for i in range(dets.shape[2]):
    if dets[0,0,i,2]>0.5:
        box = dets[0,0,i,3:7]*[w,h,w,h]; x1,y1,x2,y2 = box.astype(int)
        cv2.rectangle(img,(x1,y1),(x2,y2),(0,255,0),2)
cv2_imshow(img); cv2.waitKey(0); cv2.destroyAllWindows()
```

OUTPUT:



RESULT:

Thus, Face Detection on available online human face image datasets was implemented successfully.

EXPT NO: 10	OBJECT RECOGNITION
EXPT NO: 03/10/2025	

AIM:

To implement the Object Recognition on available online image datasets.

ALGORITHM:

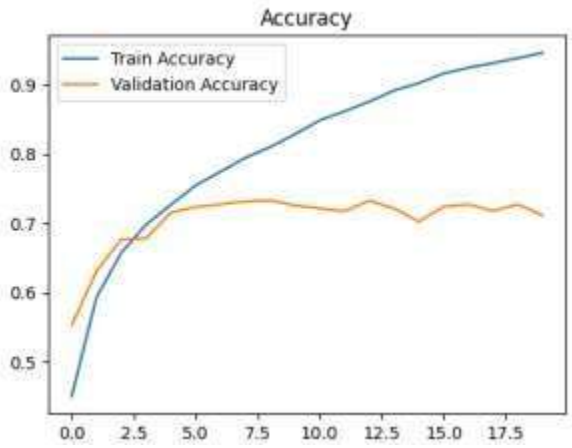
1. Load pretrained **CNN model (e.g., ResNet, MobileNet).**
2. Read and preprocess **input image** (resize, normalize).
3. Pass image through model for **prediction**.
4. Obtain top predicted labels and confidence scores.
5. Display recognized object with label.
6. Compare performance on dataset images.

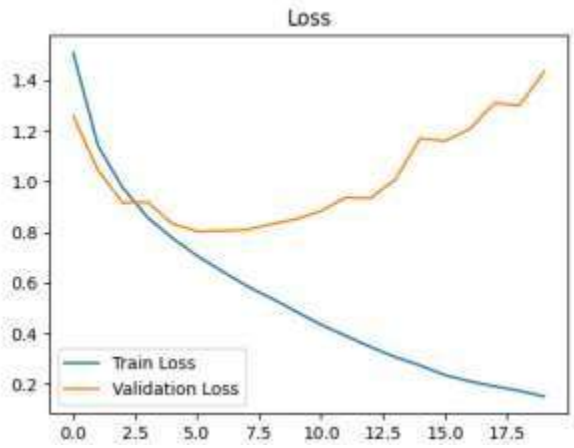
CODE:

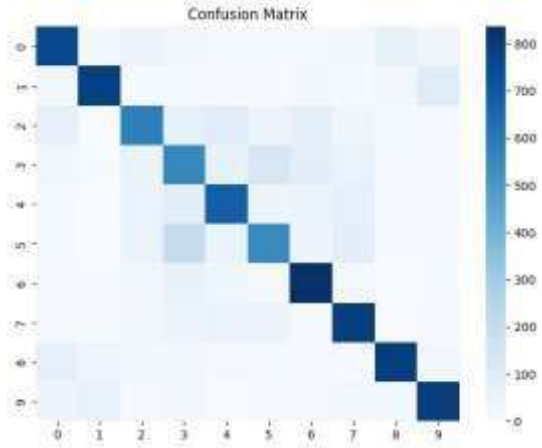
```
# Import libraries

import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

# Step 1: Load and preprocess dataset
(x_train, y_train), (x_test, y_test) = datasets.cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0  # Normalize
y_train, y_test = y_train.flatten(), y_test.flatten()

# Step 2: Define CNN model
model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(32,32,3)),
    layers.MaxPooling2D((2,2)),
```

```
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(128, (3,3), activation='relu'),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])

# Step 3: Compile model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Step 4: Train model
history = model.fit(x_train, y_train, epochs=20,
                    validation_data=(x_test, y_test),
                    batch_size=64)

# Step 5: Evaluate model
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f"\nTest Accuracy: {test_acc*100:.2f}%")

# Step 6: Plot accuracy and loss curves
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend(); plt.title("Accuracy")

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend(); plt.title("Loss")
plt.show()

# Step 7: Confusion Matrix
y_pred = np.argmax(model.predict(x_test), axis=-1)
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=False, cmap='Blues')
plt.title("Confusion Matrix")
plt.show()

# Step 8: Classification report
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

OUTPUT:







Classification Report:

	precision	recall	f1-score	support
0	0.72	0.75	0.74	1000
1	0.86	0.78	0.82	1000
2	0.67	0.58	0.62	1000
3	0.50	0.56	0.53	1000
4	0.67	0.68	0.68	1000
5	0.65	0.55	0.60	1000
6	0.73	0.84	0.78	1000
7	0.71	0.79	0.75	1000
8	0.84	0.79	0.82	1000
9	0.79	0.79	0.79	1000
accuracy			0.71	10000
macro avg	0.71	0.71	0.71	10000
weighted avg	0.71	0.71	0.71	10000

RESULT:

Thus, Object Recognition on available online image datasets was implemented successfully.