**# simple image processing**

**Import cv2**

**From google.colab.patches import cv2\_imshow Image=cv2.imread(‘image.jpg’)**

**cv2\_imshow(image)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

**gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY) cv2\_imshow(gray\_image)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

**resized\_image = cv2.resize(image,(200,200)) cv2\_imshow(resized\_image)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

**blurred\_image = cv2.GaussianBlur(image, (15, 15), 0) cv2\_imshow(blurred\_image)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

**edges = cv2.Canny(gray\_image, 100, 200)**

**cv2\_imshow(edges)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

**cv2.rectangle(image, (50,50), (300,300), (255, 0, 0), 2)**

**cv2.line(image, (60,60), (300,300), (0, 0, 255), 2)**

**cv2\_imshow(image)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

**#vehicle detection**

**import cv2**

**import time**

**import numpy as np**

**from google.colab.patches import cv2\_imshow**

**# Load the pre-trained car classifier**

**car\_classifier = cv2.CascadeClassifier('cars.xml')**

**# Capture video from file**

**cap = cv2.VideoCapture('video.avi')**

**while cap.isOpened():**

**time.sleep(0.05)  # Add a delay to simulate real-time video processing**

**ret, frame = cap.read()**

**if not ret:**

**break  # Exit the loop if no frame is captured**

**# Convert each frame to grayscale**

**gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)**

**# Detect cars in the frame**

**cars = car\_classifier.detectMultiScale(gray, 1.4, 2)**

**# Draw rectangles around detected cars**

**for (x, y, w, h) in cars:**

**cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 255), 2)**

**# Display the frame with detected cars**

**cv2\_imshow(frame)**

**# Break the loop if 'Enter' key is pressed**

**if cv2.waitKey(1) == 13:**

**break**

**# Release the video capture object and close all OpenCV windows**

**cap.release()**

**cv2.destroyAllWindows()**

**#text extraction**

**!apt-get install tesseract-ocr -y**

**!pip install pytesseract**

**import cv2**

**import pytesseract**

**# Specify the image path**

**image\_path = "text.jpg"**

**# Load the image**

**image = cv2.imread(image\_path)**

**# Convert the image to grayscale**

**gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)**

**# Extract text from the grayscale image using Tesseract**

**text = pytesseract.image\_to\_string(gray)**

**# Print the extracted text**

**print("Extracted Text:")**

**print(text)**

**#family photo face detection**

**import cv2**

**from google.colab.patches import cv2\_imshow**

**# Load the image**

**image = cv2.imread('family.jpg')**

**# Convert the image to grayscale**

**gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)**

**# Load the pre-trained face detection model (Haar Cascade)**

**classifier = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml')**

**# Detect faces in the image**

**cont = classifier.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))**

**# Draw rectangles around the detected faces**

**for x, y, w, h in cont:**

**cv2.rectangle(image, (x, y), (x+w, y+h), (0, 0, 255), 2)**

**# Display the image with detected faces**

**cv2\_imshow(image)**

**#image captioning**

**from transformers import BlipProcessor, BlipForConditionalGeneration**

**from PIL import Image**

**import matplotlib.pyplot as plt**

**image\_path = 'image.jpg' # Replace with your local image path**

**image = Image.open(image\_path).convert("RGB")**

**processor = BlipProcessor.from\_pretrained("Salesforce/blip-image-captioning-base")**

**model = BlipForConditionalGeneration.from\_pretrained("Salesforce/blip-image-captioning-base")**

**inputs = processor(images=image, return\_tensors="pt")**

**caption = processor.decode(model.generate(\*\*inputs)[0], skip\_special\_tokens=True)**

**plt.imshow(image)**

**plt.title(caption)**

**plt.axis('off')**

**plt.show()**

**#qr code**

**!apt-get install libzbar0**

**!pip install pyzbar**

**import cv2**

**from pyzbar.pyzbar import decode**

**from google.colab.patches import cv2\_imshow**

**import numpy as np**

**# Load the image containing the QR code**

**frame = cv2.imread('qr.jpg')**

**# Check if the image was loaded successfully**

**if frame is not None:**

**for qr in decode(frame):**

**# Draw the bounding box around the detected QR code**

**cv2.polylines(frame, [np.array(qr.polygon, dtype=np.int32)], True, (255, 0, 0), 3)**

**# Decode and print the QR code data**

**qr\_data = qr.data.decode('utf-8')**

**print("QR Code detected:", qr\_data)**

**# Display the image with the QR code highlighted**

**cv2\_imshow(frame)**

**cv2.waitKey(0)**

**# Clean up windows**

**cv2.destroyAllWindows()**

**else:**

**print("Error: Could not load image.")**

**import cv2**

**import numpy as np**

**from google.colab.patches import cv2\_imshow**

**# Load the image**

**image = cv2.imread('car.jpg')**

**# Check if the image was loaded successfully**

**if image is not None:**

**# Convert the image to grayscale**

**gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)**

**# Detect edges in the image using the Canny edge detector**

**edges = cv2.Canny(gray, 150, 255)**

**# Apply a binary threshold to the grayscale image**

**ret, thresh = cv2.threshold(gray, 150, 255, cv2.THRESH\_BINARY)**

**# Find contours in the thresholded image**

**contours, \_ = cv2.findContours(thresh, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_NONE)**

**# Create a copy of the original image to draw the contours on**

**contour\_image = image.copy()**

**# Draw the contours on the image**

**cv2.drawContours(contour\_image, contours, -1, (0, 255, 0), 2)**

**# Display the image with contours**

**cv2\_imshow(contour\_image)**

**# Display the thresholded image**

**cv2\_imshow(thresh)**

**# Display the edges image**

**cv2\_imshow(edges)**

**else:**

**print("Error: Could not load image.")**

|  |  |
| --- | --- |
|  | |
|  | **# Road Lane** |
| **import cv2**  **from google.colab.patches import cv2\_imshow**  **import numpy as np**  **import time**  **cap = cv2.VideoCapture('test2.mp4') while cap.isOpened():**  **time.sleep(0.05)**  **ret, frame = cap.read() if not ret:**  **break**  **gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY) edges = cv2.Canny(gray, 50, 150)**  **lines = cv2.HoughLinesP(edges, 1, np.pi/180, 50, minLineLength=300, maxLineGap=10)**  **if lines is not None:**  **for line in lines:**  **x1, y1, x2, y2 = line[0]**  **cv2.line(frame, (x1, y1), (x2, y2), (0, 255, 0), 5) cv2\_imshow(frame)**  **cap.release()**  **cv2.destroyAllWindows()** | |

**#2D-3D**

**import cv2**

**import numpy as np**

**from google.colab.patches import cv2\_imshow**

**# Load the images**

**img = cv2.imread('cube1.jpeg', cv2.IMREAD\_UNCHANGED)**

**dep = cv2.imread('cube2.jpeg', cv2.IMREAD\_GRAYSCALE)**

**# Normalize depth values and convert to integer**

**delta = (dep / 255.0 \* 10).astype(int)**

**# Initialize an array of the same shape as img**

**arr = np.zeros\_like(img)**

**# Apply depth-based shift to the image**

**for y in range(img.shape[0]):**

**for x in range(img.shape[1]):**

**dx = delta[y, x]**

**if 0 <= x + dx < img.shape[1]:**

**arr[y, x + dx] = img[y, x]**

**# Display the manipulated image**

**cv2\_imshow(arr)**

**# Wait for a key press and close any OpenCV windows**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

**#Basic Motion Detection**

**import cv2**

**import numpy as np**

**from google.colab.patches import cv2\_imshow cap=cv2.VideoCapture('testing.mp4') sub=cv2.BackgroundSubtractorMOG2()**

**while True: ret,frame=cap.read() mask=sub.apply(frame)**

**contour=cv2.findContours(mask,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE) for i in contour:**

**if cv2.contourArea(i)>550: x,y,w,h=cv2.boundingRect(i)**

**cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2) cv2\_imshow(frame)**

**cv2\_imshow(mask)**

**# Release resources cap.release() cv2.destroyAllWindows()**

**#people counting**

**import cv2**

**from google.colab.patches import cv2\_imshow**

**import numpy as np**

**import time**

**# Load the pre-trained people classifier**

**cls = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_fullbody.xml')**

**# Open the video file**

**cap = cv2.VideoCapture("peoplebg.mp4")**

**# Initialize the people count**

**ct = 0**

**while cap.isOpened():**

**time.sleep(0.05) # Pause to match video frame rate**

**# Read a frame from the video**

**ret, frame = cap.read()**

**# Break the loop if the frame was not read successfully**

**if not ret:**

**break**

**# Convert the frame to grayscale**

**gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)**

**# Detect people in the grayscale frame**

**class2 = cls.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=3)**

**# Draw rectangles around detected people and count them**

**for (x, y, w, h) in class2:**

**cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 3)**

**ct += 1**

**# Display the frame with detected people**

**cv2\_imshow(frame)**

**# Print the count of people detected in the current frame**

**print(f"People detected in this frame: {ct}")**

**# Release the video capture and close any OpenCV windows**

**cap.release()**

**cv2.destroyAllWindows()**

**# Vehicle counting**

**import cv2**

**from google.colab.patches import cv2\_imshow**

**import time**

**import numpy as np**

**# Open the video file**

**cap = cv2.VideoCapture("cars.avi")**

**# Load the pre-trained vehicle classifier**

**cls = cv2.CascadeClassifier("cars.xml")**

**# Initialize vehicle count**

**ctr = 0**

**while cap.isOpened():**

**time.sleep(0.05) # Pause to match video frame rate**

**# Read a frame from the video**

**ret, frame = cap.read()**

**# Break the loop if the frame was not read successfully**

**if not ret:**

**break**

**# Convert the frame to grayscale**

**gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)**

**# Detect vehicles in the grayscale frame**

**class1 = cls.detectMultiScale(gray, scaleFactor=1.4, minNeighbors=2)**

**# Draw rectangles around detected vehicles and count them**

**for x, y, w, h in class1:**

**cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)**

**ctr += 1**

**# Display the frame with detected vehicles**

**cv2\_imshow(frame)**

**# Release the video capture and close any OpenCV windows**

**cap.release()**

**cv2.destroyAllWindows()**

**# Print the total count of detected vehicles**

**print(f"Total number of vehicles detected: {ctr}")**

**#emotion Detection**

**import cv2**

**from deepface import DeepFace**

**from google.colab.patches import cv2\_imshow**

**# Load the image**

**image = cv2.imread('emotion.jpeg')**

**# Analyze the image for emotions**

**emotion = DeepFace.analyze(image, actions=['emotion'])**

**# Display the image with emotion analysis**

**cv2\_imshow(image)**

**# Print the emotion analysis result**

**print(emotion)**

**#region segmentation**

**import numpy as np**

**import matplotlib.pyplot as plt from skimage.feature import canny**

**from skimage.color import rgb2gray,label2rgb from skimage.filters import sobel**

**from skimage import data, morphology**

**from skimage.segmentation import watershed import scipy.ndimage as nd**

**img=rgb2gray(data.rocket()) ed=canny(img) fill=nd.binary\_fill\_holes(ed) map=sobel(img)**

**mark=np.zeros\_like(image,dtype=int) mark[image<0.2]=1**

**mark[image>0.6]=2**

**seg=watershed(map,mark) seg\_fill=nd.binary\_fill\_holes(seg-1) lab,\_=nd.label(seg\_fill) img1=label2rgb(lab,image=image)**

**plt.imshow(img) plt.imshow(seg) plt.imshow(ed) plt.imshow(img1)**