CV Programs

Part-A

Write a program in python to perform various basic image processing operations: Reading image, writing image, conversion of images, and complement of an image using OpenCV.

```
#Implementing various basic image processing operation
#Reading,writing,conversion, and complement of an image
```

[1]

Generate | + Code | + Markdown

```
import cv2
```

[1]

```
image=cv2.imread("images.jpeg")
if image is not None:
    cv2.imshow('image',image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
else:
    print("Image not found")
cv2.imwrite('output.jpeg',image)
```

[2]

```
True
```

```
img=cv2.imread('images.jpeg')
gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
cv2.imshow('image',gray)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[3]

```
image=cv2.imread("images.jpeg")
complement=255-image
cv2.imshow('Complement image',complement)
cv2.waitKey(0)
cv2.destroyAllWindows()
cv2.imwrite('Complement.jpg',complement)
```

[4]

```
True
```

Write a program in python to read and display the videos using OpenCV

```
#Python Programm to read and display a video
```

[1]

```
import cv2
cap=cv2.VideoCapture(0)
while cap.isOpened():
    ret,frame=cap.read()
    if not ret:
        break
    cv2.imshow('Video',frame)
    if cv2.waitKey(1) & 0xFF==ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

[2]

Write a program in python to resize, crop and rotate an image using OpenCV.

```
import cv2
import numpy as np


image_path='face_eye_detection.jpg'
image=cv2.imread(image_path)


resized_image=cv2.resize(image,(300,300))


h,w,_=resized_image.shape
start_x,start_y=(w-200)//2,(h-200)//2
cropped_image=resized_image[start_y:start_y+200,start_x:start_x+200]


n,w=cropped_image.shape[:2]
center=(w//2,h//2)
rotation_matrix=cv2.getRotationMatrix2D(center,45,1.0)
rotation_image=cv2.warpAffine(cropped_image,rotation_matrix,(w,h))


cv2.imshow('orgimag',image)
cv2.imshow('resize',resized_image)
cv2.imshow('cropped',cropped_image)
cv2.imshow('rotated',rotation_image)


cv2.waitKey(0)
cv2.destroyAllWindows()
```

Write a program in python to implement the various image filtering mechanisms using OpenCV

```
[1]
import cv2
import numpy as np
input_image=cv2.imread('images.jpeg',cv2.IMREAD_COLOR)

[2]
mean_filtered=cv2.blur(input_image,(3,3))

[3]
median_filtered=cv2.medianBlur(input_image,3)

[4]
laplacian=cv2.Laplacian(input_image,cv2.CV_64F)
laplacian_filtered=cv2.convertScaleAbs(laplacian)

[5]
cv2.imshow('Original Colorful Image',input_image)
cv2.imshow('Mean Filtered Image',mean_filtered)
cv2.imshow('Median Image',median_filtered)
cv2.imshow('Laplacian edge Enchancement Image',laplacian_filtered)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Write a program in python to demonstrate Bitwise Operations on Binary Images using OpenCV.

```
[4]
import cv2
```

```
[5]
img1=cv2.imread('input1.png')
img2=cv2.imread('input2.png')
```

```
[6]
dest_and=cv2.bitwise_and(img2,img1,mask=None)
dest_or=cv2.bitwise_or(img2,img1,mask=None)
dest_not1=cv2.bitwise_not(img2,img1,mask=None)
dest_not2=cv2.bitwise_not(img2,img1,mask=None)
dest_xor=cv2.bitwise_xor(img2,img1,mask=None)
```

```
[9]
cv2.imshow('Image 1',img1)
cv2.imshow('Image 2',img2)

cv2.imshow('AND',dest_and)
cv2.imshow('OR',dest_or)
cv2.imshow('NOT1',dest_not1)
cv2.imshow('NOT2',dest_not2)
cv2.imshow('XOR',dest_xor)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Write a program in python to Draw different geometric shapes and to write text on images using OpenCV.

```
[2]
import numpy as np
import cv2
```

```
[3]
Img=np.zeros((512,512,3),dtype='uint8')
```

```
[4]
Img=cv2.line(Img,(0,0),(100,100),(255,255,255),4)
```

```
[5]
Img=cv2.arrowedLine(Img,(0,50),(100,150),(255,255,255),4)
Img=cv2.ellipse(Img,(300,100),(100,50),30,0,360,(255,0,0),-1)
Img=cv2.circle(Img,(450,100),30,(255,0,0),4)
Img=cv2.rectangle(Img,(20,200),(200,300),(0,255,0),2)
```

```
[6]
font=cv2.FONT_HERSHEY_SIMPLEX
org=(50,400)
fontScale=2
color=(0,0,255)
thickness=3
Img=cv2.putText(Img,'OpenCV',org,font,fontScale,color,thickness,cv2.LINE_AA,False)
Img=cv2.putText(Img,'OpenCV',org,font,fontScale,color,thickness,cv2.LINE_AA,True)
```

```
cv2.imshow('Image',Img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Write a program in python to perform different Morphological operations on images based on OpenCV

```
[1]
import cv2
import numpy as np

[2]
img=cv2.imread('morph.jpg')
kernel=np.zeros((5,5),np.uint8)

[3]
img_erosion=cv2.erode(img,kernel,iterations=1)
img_dilation=cv2.dilate(img,kernel,iterations=1)
img_opening=cv2.morphologyEx(img,cv2.MORPH_OPEN,kernel,iterations=1)
img_closing=cv2.morphologyEx(img,cv2.MORPH_CLOSE,kernel,iterations=1)
img_morph_gradient=cv2.morphologyEx(img,cv2.MORPH_GRADIENT,kernel)

[4]
cv2.imshow('Original',img)
cv2.imshow('Erosion',img_erosion)
cv2.imshow('Dilation',img_dilation)
cv2.imshow('Opening',img_opening)
cv2.imshow('Closing',img_closing)
cv2.imshow('Morphological Gradient',img_morph_gradient)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Implement different Thresholding techniques on images using openCV.

```
import cv2
import numpy as np
image1=cv2.imread("tiger.jpg")
img=cv2.cvtColor(image1,cv2.COLOR_BGR2GRAY)
ret,thresh1=cv2.threshold(img,127,255,cv2.THRESH_BINARY)
ret,thresh2=cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)
ret,thresh3=cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
ret,thresh4=cv2.threshold(img,127,255,cv2.THRESH_TOZERO)
ret,thresh5=cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)

cv2.imshow("Original Image",img)
cv2.imshow("Binary threshold",thresh1)
cv2.imshow("Binarythreshold inverted",thresh2)
cv2.imshow("Trunc",thresh3)
cv2.imshow("Tozero",thresh4)
cv2.imshow("Tozero_INV",thresh5)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Implement Edge detection and Contour detection on images using openCV.

```
import cv2
image = cv2.imread('images.jpeg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(gray, (3, 3), 0)

wide_Edge = cv2.Canny(blurred, 10, 100)
Mid_Edge = cv2.Canny(blurred, 30, 150)
tight_Edge = cv2.Canny(blurred, 240, 250)

cv2.imshow('Original', image)
cv2.imshow('Wide Edge Map', wide_Edge)
cv2.imshow('Mid Edge Map', Mid_Edge)
cv2.imshow('Tight Edge Map', tight_Edge)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
contours,_=cv2.findContours(Mid_Edge,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
image_copy=image.copy()
cv2.drawContours(image_copy,contours,-1,(0,255,0),3)
print('Number of contours found=' + str(len(contours)))
cv2.imshow('Contours',image_copy)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
Number of contours found=83
```

Part-B

Write a program in python to demonstrate image enhancement using contrast adjustment and histogram equalization using OpenCV.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def adjust_contrast(image,alpha,beta):
    adjusted_image=cv2.convertScaleAbs(image,alpha=alpha,beta=beta)
    return adjusted_image
def equalize_histogram(image):
    equalized_image=cv2.equalizeHist(image)
    return equalized_image
input_image=cv2.imread('images.jpeg',cv2.IMREAD_GRAYSCALE)
alpha=1.5
beta=0
contrast_adjusted_image=adjust_contrast(input_image,alpha,beta)
equalized_image=equalize_histogram(input_image)
cv2.imshow('Orginal Image',input_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
cv2.imshow('Contrast Adjusted Image',contrast_adjusted_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
cv2.imshow('Equalized Image',equalized_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
cv2.imwrite('contrast_adjusted_output.jpeg',contrast_adjusted_image)
cv2.imwrite('equalized_output.jpeg',equalized_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[2]

```
plt.figure(figsize=(12,6))
plt.subplot(2,2,1)
plt.hist(input_image.ravel(),256,[0,256])
plt.title('Original Image Histogram')
plt.subplot(2,2,2)
plt.hist(contrast_adjusted_image.ravel(),256,[0,256])
plt.title('Contrast Adjusted Image Histogram')
plt.subplot(2,2,3)
plt.hist(equalized_image.ravel(),256,[0,256])
plt.title('Equalized Image Histogram')
```

[4]

Demonstrate Haar feature-based cascade classifiers for Face and Eye Detection on images.

```
import cv2
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades+'haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades+'haarcascade_eye.xml')


image=cv2.imread('face_eye_detection.jpg')
gray=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)


faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))


for (x,y,w,h) in faces:
    cv2.rectangle(image,(x,y),(x+w,y+h),(255,0,0),2)
    face_roi = gray[y:y+h, x:x+w]
    color_roi_color = image[y:y+h, x:x+w]
    eyes = eye_cascade.detectMultiScale(face_roi)
    for (ex,ey,ew,eh) in eyes:
        cv2.rectangle(color_roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)
    noses = nose_cascade.detectMultiScale(face_roi)
    for (nx,ny,nw,nh) in noses:
        cv2.rectangle(color_roi_color,(nx,ny),(nx+nw,ny+nh),(0,0,255),2)


import cv2

nose_cascade = cv2.CascadeClassifier('haarcascade_mcs_nose.xml')  # Ensure correct path

if nose_cascade.empty():
    print("Error: Nose cascade not loaded. Check the file path!")


cv2.imshow('Face and Eye Detection',image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Write a program in python to demonstrate Handwritten Digit Recognition on MNIST dataset.

```
[11]
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt

[12]
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

[13]
train_images, test_images=train_images/255.0, test_images/255.0

[14]
train_labels=to_categorical(train_labels)
test_labels=to_categorical(test_labels)

Empty markdown cell, double-click or press enter to edit.

model=models.Sequential()
model.add(layers.Flatten(input_shape=(28,28)))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(10, activation='softmax'))
```

```
[16]
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

[17]
model.fit(train_images, train_labels, epochs=5)
```

```
test_loss, test_acc=model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```
[18]

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9716 - loss: 0.0887
Test accuracy: 0.9757000207901001
```

```
predictions=model.predict(test_images)
```
[19]

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```

Generate

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(test_images[i], cmap=plt.cm.binary)
    plt.xlabel('Predicted: '+str(predictions[i].argmax()))
    true_label=test_labels[i].argmax()
    color='blue' if true_label==predictions[i].argmax() else 'red'
    plt.xlabel('Predicted: '+str(predictions[i].argmax()), color=color)
plt.show()
```
[20]

Develop a classification model using YOLO object detection algorithm using OpenCV.

```
import cv2
import numpy as np

net=cv2.dnn.readNet("yolov3.weights","yolov3.cfg")
layer_names=net.getLayerNames()
output_layers=[layer_names[i-1] for i in net.getUnconnectedOutLayers()]
```
[1]

```
with open("coco.names","r") as f:
    classes=[line.strip() for line in f.readlines()]
```
[2]

```
Image=cv2.imread("1.webp")
height,width=Image.shape[:2]
```
[3]

Generate + Code + Markdown

```
blob=cv2.dnn.blobFromImage(Image,0.00392,(416,416),(0,0,0),True,crop=False)
net.setInput(blob)
outs=net.forward(output_layers)
```
[4]

```
detected_boxes=[]
detected_classes=[]
detected_confidences=[]
```
[5]

```
for out in outs:
    for detection in out:
        scores=detection[5:]
        class_id=np.argmax(scores)
        confidence=scores[class_id]
        if confidence>0.2:
            center_x,center_y,width,height=(detection[0:4]*np.array([width,height,width,height])).astype("int")
            x=int(center_x-width/2)
            y=int(center_y-height/2)
            detected_boxes.append([x,y,int(width),int(height)])
            detected_classes.append(class_id)
            detected_confidences.append(float(confidence))
```
[6]

```
indices=cv2.dnn.NMSBoxes(detected_boxes,detected_confidences,0.5,0.3)
print(indices)
for i in indices.flatten():
    x,y,w,h=detected_boxes[i]
    label=f"{classes[detected_classes[i]]}:{detected_confidences[i]:.2f}"
    cv2.rectangle(Image,(x,y),(x+w,y+h),(0,255,0),2)
    cv2.putText(Image,classes[detected_classes[i]],(x,y-5),cv2.FONT_HERSHEY_SIMPLEX,0.5,(0,255,0),2)
```
[7]

```
[1]
```

```
cv2.imshow("YOLO Object Detection",Image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```
[8]

Write a Python program to demonstrate image data augmentation techniques.

```
import tensorflow as tf
from tensorflow.keras import layers
import matplotlib.pyplot as plt
import tensorflow_datasets as tfds
```

```
(train_ds, val_ds, test_ds), metadata = tfds.load(
    'tf_flowers',
    split=['train[:80%]', 'train[80%:90%]', 'train[90%:]'],
    with_info=True,
    as_supervised=True,
)
BATCH_SIZE = 32
IMG_SIZE = (180, 180)

def preprocess_image(image, label):
    image = tf.image.resize(image, IMG_SIZE)
    image = tf.cast(image, tf.float32)
    image = image/255.0
    return image, label
train_ds= train_ds.map(preprocess_image,num_parallel_calls=tf.data.AUTOTUNE)
val_ds= val_ds.map(preprocess_image,num_parallel_calls=tf.data.AUTOTUNE)
test_ds= test_ds.map(preprocess_image,num_parallel_calls=tf.data.AUTOTUNE)
```

```
data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.2),
    layers.RandomZoom(0.2),
    layers.RandomContrast(0.2),
])
```

```
def plot_images(original, augmented):
    plt.figure(figsize=(10, 10))
    for i in range(len(original)):
        plt.subplot(2, len(original), i + 1)
        plt.imshow(original[i])
        plt.title("Original")
        plt.axis('off')

        plt.subplot(2, len(augmented), len(original) + i + 1)
        plt.imshow(augmented[i])
        plt.title("Augmented")
        plt.axis('off')
    plt.tight_layout()
    plt.show()
```

```
num_images=5
sample_images=[]
for image,_ in train_ds.take(num_images):
    sample_images.append(image)
augmented_images=data_augmentation(tf.convert_to_tensor(sample_images),training=True)
plot_images(sample_images,augmented_images.numpy())
```

Write a Python Program to demonstrate feature extraction and t-SNE Visualization.

```
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications.resnet50 import preprocess_input
from sklearn.manifold import TSNE
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
(x_train, y_train), (_,_)=tf.keras.datasets.cifar10.load_data()
```

```
num_images=100
images=x_train[:num_images]
labels=y_train[:num_images].flatten()
```

```
def preprocess_images(images):
    images_resized=np.array([tf.image.resize(image, [224,224]).numpy() for image in images])
    return preprocess_input(images_resized)
images=preprocess_images(images)
```

```
resnet_model=ResNet50(weights='imagenet', include_top=False,pooling='avg')
```

```
def extract_features(images):
    return resnet_model.predict(images)
features=extract_features(images)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 7s 1s/step
```

```
tsne=TSNE(n_components=2, random_state=42)
features_2D=tsne.fit_transform(features)
```

```
cifar10_classes=[
    "airplae",
    "automobile",
    "bird",
    "cat",
    "deer",
    "dog",
    "frog",
    "horse",
    "ship",
    "truck"
]
```

```
labels_named=[cifar10_classes[label] for label in labels]
```

```
plt.figure(figsize=(12,7))
sns.scatterplot(x=features_2D[:,0], y=features_2D[:,1], hue=labels_named, palette="tab10",s=80,alpha=0.8)
plt.title("t-SNE visualization of ResNet50 features")
plt.xlabel("t-SNE Component1")
plt.ylabel('t-sNE Component 2')
plt.legend(loc="best", bbox_to_anchor=(1,1), title="Class Labels")
plt.show()
```

Write a Python Program to implement reverse image search engine using keras.

```
import tensorflow as tf
import tensorflow_datasets as tfds
import numpy as np
import cv2
import matplotlib.pyplot as plt
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.preprocessing import image
from scipy.spatial import distance
from tqdm import tqdm
```

```
model=VGG16(weights='imagenet',include_top=False,input_shape=(224,224,3))
dataset_name='caltech101'
dataset,info=tfds.load(name=dataset_name,as_supervised=True,with_info=True)
image_data=[]
features_db={}
```

```
for img_tensor, label in tqdm(dataset['train'].take(500)):
    img_array = tf.image.resize(img_tensor, (224, 224)).numpy()
    img_array = np.expand_dims(img_array, axis=0)
    img_array = preprocess_input(img_array)
    features = model.predict(img_array).flatten()

    image_data.append((img_tensor, features, label.numpy()))
image_data = np.array(image_data, dtype=object)
```

```
def search_similar_images(query_index,top_n=5):
    query_features=image_data[query_index][1]
    distances={}

    for i,(_,features,_) in enumerate(image_data):
        dist=distance.euclidean(query_features,features)
        distances[i]=dist

    sorted_distances=sorted(distances.items(),key=lambda x:x[1])[:top_n]

    plt.figure(figsize=(15,5))
    plt.subplot(1,top_n+1,1)
    plt.imshow(image_data[query_index][0])
    plt.axis('off')
    plt.title("Query Image")

    for i, (img_idx, similarity) in enumerate(sorted_distances):
        plt.subplot(1,top_n+1,i+2)
        plt.imshow(image_data[img_idx][0])
        plt.axis('off')
        plt.title(f"Dist: {similarity:.2f}")
    plt.show()
```

```
query_index=10
search_similar_images(query_index,top_n=5)
```