UN5550

INTRODUCTION TO DATA SCIENCE

FINAL PROJECT REPORT

ON

OPENCV

TEAM ORANGE

HARANADH REDDY RAVI, BALAJI MEDIBOINA,
BHAVANI CHALAMALLA

UNDER THE GUIDANCE OF

DR DUKKA KC

Michigan Technological University

Table Of Contents:

RESULTS & CONCLUSION

Michigan Technological University

Introduction:

Computer Vision: Computer vision is a field of artificial intelligence that deals with enabling machines to interpret and understand visual information from the world around them. This involves developing algorithms and techniques that can recognize patterns and objects in images and videos, and extract meaningful information from them. Practical applications of computer vision include robotics, surveillance, healthcare, automotive, and entertainment. OpenCV is a powerful library that provides tools and techniques for computer vision applications.

OpenCV: OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. It was developed by Intel in 1999 and later released as an open-source project in 2000. OpenCV is written in C++, python and offers a range of computer vision and machine learning algorithms, tools, and techniques, making it a powerful and versatile tool for developing computer vision applications.

OpenCV is widely used in various industries such as robotics, automotive, healthcare, entertainment, and security, to name a few. It has become a standard library for computer vision researchers and practitioners around the world, and its popularity continues to grow with the advancement of artificial intelligence and machine learning.

In this tutorial, we will introduce OpenCV and its various features, including image processing, object detection, and face recognition. We will also provide examples of how to use OpenCV in Python, a popular programming language for data science and machine learning. By the end of this tutorial, you will have a basic understanding of OpenCV and how to use it to solve real-world problems in computer vision.

Background and Related Work:

OpenCV was first developed by Intel in 1999 as a library of computer vision algorithms optimized for Intel CPUs. In 2000, Intel released OpenCV as an open-source project, and since then it has grown into a widely-used and respected library in the computer vision community.

Over the years, OpenCV has been used in a variety of research projects and commercial applications, including robotics, augmented reality, and facial recognition. Its popularity has also led to the development of numerous resources and libraries that build on top of OpenCV, such as OpenFace, a toolkit for facial recognition, and OpenPose, a toolkit for human pose estimation.

In addition to OpenCV, there are other libraries and frameworks for computer vision and image processing, such as MATLAB, TensorFlow, and PyTorch. However, OpenCV remains a popular choice for many computer vision applications due to its robustness, versatility, and ease of use.

Design of Approach/Methodology:

Installing OpenCV: The first step is to install the library and its dependencies on your machine. This may involve downloading and installing the appropriate packages for your operating system, or building the library from source.

Image Processing: Once OpenCV is installed, you can begin to work with images and perform basic image processing tasks such as loading, displaying, and saving images.

Basic Image Processing Operations: This may involve tasks such as resizing, cropping, and rotating images. It also involves Image filtering(OpenCV provides a range of image filtering techniques, such as blurring, sharpening, and edge detection. These techniques can be used to enhance the quality of images and extract useful information from them.) and thresholding.

Object Detection: OpenCV includes a range of algorithms for detecting and recognizing objects in images and videos, such as Haar cascades and deep learning-based methods.

Feature Detection and Description: OpenCV provides tools for detecting and describing features in images, such as corners and blobs. These features can be used for tasks such as car & pedestrian detection. Also for face & eyes recognition.

Machine Learning: OpenCV also includes tools and libraries for machine learning, such as support vector machines and decision trees. These techniques can be used for tasks such as image segmentation, Optical character recognition (OCR) and Gesture recognition.

Deep Learning with OpenCV: OpenCV provides support for popular Deep Learning frameworks like TensorFlow, PyTorch, and Caffe. Some examples of Deep Learning tasks that can be performed using OpenCV include:

- Image and video classification
- Semantic segmentation
- Generative models
- Neural style transfer

Real world applications: Finally, you can use your knowledge of OpenCV to develop real-world applications, such as

- **Autonomous Vehicles:**
  Object detection and recognition: Autonomous vehicles use cameras to capture images and videos of the surrounding environment. OpenCV can be used to detect and recognize objects in these images, such as cars, pedestrians, traffic signs, and traffic lights. This helps the vehicle to navigate and make decisions based on the environment.

  Lane detection and tracking: OpenCV can also be used to detect and track lanes on the road. This is important for the vehicle to maintain its position on the road and avoid collisions.

  Optical flow analysis: OpenCV can be used to analyze the motion of objects in the environment. This can help the vehicle to predict the motion of other vehicles, pedestrians, and objects in the environment.

  Augmented reality: OpenCV can be used to create augmented reality experiences for the passengers in the vehicle. This can include displaying information about the environment, such as the speed limit, nearby points of interest, and weather conditions.

  Overall, it can help the vehicle to navigate, make decisions, and provide a safe and comfortable experience for the passengers.

- **Medical Imaging:**
  Segmentation: OpenCV can be used to segment medical images into different regions or structures. This can help to identify and isolate specific areas of interest such as tumors, blood vessels, or organs.

  Feature extraction: OpenCV can be used to extract features from medical images such as texture, shape, and color. These features can be used to identify patterns or abnormalities in the images that may be indicative of certain medical conditions.

  Image registration: OpenCV can be used to align and register multiple medical images of the same patient, taken at different times or using different imaging modalities. This can help medical professionals to track the progression of a disease or the effectiveness of a treatment.

  Overall, OpenCV can help to improve the accuracy and efficiency of medical diagnoses and treatments, leading to better patient outcomes.

Conclusion:

Summary:

OpenCV is a powerful open-source library for computer vision that can be used for a wide range of applications, including image and video processing, object detection, and machine learning.

The library includes many advanced algorithms and techniques, such as feature detection and matching, camera calibration, and deep learning, which can be used to solve complex computer vision problems.

While OpenCV has many strengths, it also has some limitations and challenges, such as limited support for non-image data, performance and computational complexity, and limited accuracy and robustness.

To address these challenges, you can consider using other libraries or frameworks that are better suited for non-image data, optimizing your OpenCV code using parallel processing techniques or specialized hardware, using advanced algorithms or techniques to improve accuracy and robustness, and contributing to the OpenCV community to find solutions to specific problems.

Despite these challenges, OpenCV remains one of the most popular and widely used libraries for computer vision, with a strong community of developers and users who continue to push the boundaries of what is possible.

Scope for OpenCV:

OpenCV is an incredibly versatile library that has been widely adopted in both academic and industrial settings. As the field of computer vision continues to expand, OpenCV is likely to play a significant role in enabling new applications and discoveries. Some of the areas where OpenCV could be used in the future include:

Robotics: OpenCV can be used to help robots perceive and interact with their environments, enabling them to perform tasks that were previously difficult or impossible.

Agriculture: OpenCV can be utilized in agriculture to monitor crops and optimize farming practices, resulting in increased yields and reduced waste.

Security: Security systems can take advantage of OpenCV to detect and track intruders, identify faces, and analyze video footage.

Entertainment: OpenCV has potential applications in the entertainment industry, such as face tracking and gesture recognition in video games and movies for augmented and virtual reality experiences.

References:

OpenCV Documentation: This is the official documentation for OpenCV, and it contains detailed information about the library, including tutorials, examples, and API references. You can find it at https://docs.opencv.org/.

Learning OpenCV 4 Computer Vision with Python 3: This book by Joseph Howse and Prateek Joshi provides a practical guide to using OpenCV for computer vision applications, with examples and code snippets in Python. You can find it on Amazon or other online bookstores.

OpenCV Python Tutorial: This is a series of tutorials on using OpenCV with Python, created by the OpenCV community. You can find it at https://opencv-python-tutroals.readthedocs.io/en/latest/.

OpenCV-Python Tutorials: Another set of tutorials on using OpenCV with Python, created by Abid K. You can find it at https://github.com/abidrahmank/OpenCV2-Python-Tutorials.

OpenCV Computer Vision Projects with Python: This book by Joseph Howse and Prateek Joshi provides a set of projects for learning OpenCV, with step-by-step instructions and code examples. You can find it on Amazon or other online bookstores.

OpenCV for Beginners: This is a series of tutorials on using OpenCV for beginners, created by the OpenCV community. You can find it at https://www.learnopencv.com/.

OpenCV-Python Samples: This is a set of code samples demonstrating various OpenCV features and techniques, created by the OpenCV community. You can find it at https://github.com/opencv/opencv/tree/master/samples/python.

Appendix:

Source Code:

First we need to install OpenCV. For this

```
pip install opencv-python
```

Then to check the installation of OpenCV

```
import cv2
import numpy as np
```

```
# Load an image using 'imread' specifying the path to image
image = cv2.imread('C:/Users/91630/Downloads/husky.jpg',0)

# Our file 'input.jpg' is now loaded and stored in python
# as a varaible we named 'image'

# To display our image variable, we use 'imshow'
# The first parameter will be title shown on image window
# The second parameter is the image varialbe
cv2.imshow("test image", image)

# 'waitKey' allows us to input information when a image window is open
# By leaving it blank it just waits for anykey to be pressed before
# By placing numbers (except 0), we can specify a delay for how long you keep the window open (time is in milliseconds here)
cv2.waitKey()

# This closes all open windows
# Failure to place this will cause your program to hang
cv2.destroyAllWindows()
```

Shape gives the dimensions of the image array

```
image.shape   ## (height, width, channels)  ## for colourful images with RGB the channels will be 3
```

Basic image processing Operations

```
##Resizing the image
import cv2

# Load the image
image1 = cv2.imread('C:/Users/91630/Downloads/husky.jpg', 1)

# Resize the image, set the desired new image size
resized = cv2.resize(image1, (600, 500))

# Convert the resized image to grayscale
gray = cv2.cvtColor(resized, cv2.COLOR_BGR2GRAY)

# Display the original and resized images
cv2.imshow("Original Image", image1)
cv2.imshow("Resized Image", resized)
cv2.imshow("Grayscale Image", gray)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Gaussian Blur

```
import cv2

# Load the image
image4 = cv2.imread('C:/Users/91630/Downloads/img1.jpg',1)

# Display the original image
cv2.imshow('Original Image', image4)

# Apply Gaussian blur filtering
ksize = (9, 9)  # Kernel size (odd number)
sigmaX = 0      # Standard deviation along X-axis (0 means calculated based on ksize)
sigmaY = 0      # Standard deviation along Y-axis (0 means calculated based on ksize)
blurred_image = cv2.GaussianBlur(image4, ksize, sigmaX, sigmaY)

# Display the blurred image
cv2.imshow('Blurred Image', blurred_image)

# Wait for a key press and then close all open windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Thresholding

```
import cv2

# Load the image
image5 = cv2.imread('C:/Users/91630/Downloads/img1.jpg', cv2.IMREAD_GRAYSCALE)

# Display the original image
cv2.imshow('Original Image', image5)

# Apply thresholding
ret, thresh_image = cv2.threshold(image5, 127, 255, cv2.THRESH_BINARY)

# Display the thresholded image
cv2.imshow('Thresholded Image', thresh_image)

# Wait for a key press and then close all open windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

How do we save images in OpenCV?

```
cv2.imwrite('image.png',image5)
cv2.imwrite('image.jpg',image5)   ## We can save images in whichever format we want just by specifying it
```

Face Detection using HAAR Cascade Classifiers

```
import cv2
# We point OpenCV's CascadeClassifier function to where our
# classifier (XML file format) is stored
face_classifier = cv2.CascadeClassifier('C:/Users/91630/OneDrive/Documents/FP/haarcascade_frontalface_default.xml')

# Load our image then convert it to grayscale
image = cv2.imread('C:/Users/91630/OneDrive/Documents/FP/Trumph.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Our classifier returns the ROI of the detected face as a tuple
# It stores the top left coordinate and the bottom right coordiantes
faces = face_classifier.detectMultiScale(gray, 1.3, 5)

# When no faces detected, face_classifier returns and empty tuple
if faces is ():
    print("No faces found")

# We iterate through our faces array and draw a rectangle
# over each face in faces
for (x,y,w,h) in faces:
    cv2.rectangle(image, (x,y), (x+w,y+h), (100,0,200), 2)
    #cv2.circle(image, (100, 200), 50, (127, 0, 255), 2)
    cv2.imshow('Face Detection', image)
    #cv2.imshow("Image with Circle", image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Car & Pedestrian Detection

```
import cv2
import time
import numpy as np
# Create our body classifier
car_classifier =
cv2.CascadeClassifier('C:/Users/91630/OneDrive/Documents/FP/haarcascade_car.xml')
# Initiate video capture for video file
cap = cv2.VideoCapture('C:/Users/91630/OneDrive/Documents/FP/car.MP4')


# Loop once video is successfully loaded
while cap.isOpened():


    time.sleep(.05)
    # Read first frame
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)


    # Pass frame to our car classifier
    cars = car_classifier.detectMultiScale(gray, 1.4, 2)


    # Extract bounding boxes for any bodies identified
    for (x,y,w,h) in cars:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 255), 2)
        cv2.imshow('Cars', frame)
 if cv2.waitKey(1) == 13: #13 is the Enter Key
        break
cap.release()
cv2.destroyAllWindows()
```

Full body recognition (pedestrian recognition)

```
import cv2
import numpy as np


# Create our body classifier
body_classifier = cv2.CascadeClassifier('C:/Users/91630/OneDrive/Documents/FP/haarcascade_fullbody.xml')


# Initiate video capture for video file
cap = cv2.VideoCapture('C:/Users/91630/OneDrive/Documents/FP/pedestrian.MP4')


# Loop once video is successfully loaded
while cap.isOpened():


    # Read first frame
    ret, frame = cap.read()
    #frame = cv2.resize(frame, None,fx=0.5, fy=0.5, interpolation = cv2.INTER_LINEAR)


    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Pass frame to our body classifier
    bodies = body_classifier.detectMultiScale(gray, 1.2, 3)
    # Extract bounding boxes for any bodies identified
    for (x,y,w,h) in bodies:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 255), 2)
        cv2.imshow('Pedestrians', frame)
  if cv2.waitKey(1) == 13: #13 is the Enter Key
        break
cap.release()
cv2.destroyAllWindows()
```

Let's make a live face & eye detection, keeping the face in-view at all times

```
import cv2
# Loading the cascades
face_cascade = cv2.CascadeClassifier('C:/Users/91630/OneDrive/Documents/FP/haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('C:/Users/91630/OneDrive/Documents/FP/haarcascade_eye.xml')
# Defining a function that will do the detections
def detect(gray, frame):
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    for (x, y, w, h) in faces:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = frame[y:y+h, x:x+w]
        eyes = eye_cascade.detectMultiScale(roi_gray, 1.1, 3)
        for (ex, ey, ew, eh) in eyes:
            cv2.rectangle(roi_color, (ex, ey), (ex+ew, ey+eh), (0, 255, 0), 2)
    return frame
# Doing some Face Recognition with the webcam
video_capture = cv2.VideoCapture(0)
while True:
    _, frame = video_capture.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    canvas = detect(gray, frame)
    cv2.imshow('Video', canvas)
    if cv2.waitKey(1) & 0xFF == ord('k'):
        break
video_capture.release()
cv2.destroyAllWindows()
```