Assignment: OpenCV Fundamentals - Week 1

A. Introduction

OpenCV (Open Source Computer Vision Library) is one of the most widely used open-source libraries for computer vision and image processing tasks. Written in C++ with bindings for Python and other languages, it offers a comprehensive set of tools for image transformation, feature extraction, object detection, camera calibration, and deep learning integration. Its modular architecture enables developers to efficiently implement and combine different functionalities for diverse real-world applications.

The purpose of this week's assignment is to build a strong foundational understanding of OpenCV's core modules, theoretical principles, and practical usage. By exploring topics such as image manipulation, video capture, feature detection, and edge extraction, the assignment bridges theoretical image processing concepts with hands-on coding experience. The exercises, demonstrations, and research conducted during the week provide the groundwork for more advanced projects in computer vision, robotics, medical imaging, and artificial intelligence.

B. Sources Consulted

1. Introduction to OpenCV

- OpenCV Documentation – Introduction and Setup:
  `https://docs.opencv.org/4.x/d9/df8/tutorial_root.html`
- PyImageSearch – Getting Started with OpenCV:
  `https://pyimagesearch.com/2018/07/19/opencv-tutorial-a-guide-to-learn-opencv/`

2. Core Functionality (core module)

- OpenCV Docs – Core Module Overview:
  `https://docs.opencv.org/4.x/d0/de1/group__core.html`
- LearnOpenCV – Numpy Array and OpenCV Basics:
  `https://learnopencv.com/opencv-numpy-tutorial/`

3. Image Processing (imgproc module)

- OpenCV Docs – Image Processing (imgproc):
  `https://docs.opencv.org/4.x/d7/dbd/group__imgproc.html`
- GeeksforGeeks – Image Processing in OpenCV:
  `https://www.geeksforgeeks.org/image-processing-in-opencv/`
- PyImageSearch – Image Transformations and Filtering:
  `https://pyimagesearch.com/category/image-processing/`

4. Application Utils (highgui, imgcodecs, videoio modules)

- OpenCV Docs – HighGUI Module:
  `https://docs.opencv.org/4.x/d7/dfc/group__highgui.html`
- OpenCV Docs – Image and Video I/O:
  `https://docs.opencv.org/4.x/d4/da8/group__imgcodecs.html`
- GeeksforGeeks – Reading and Writing Images in OpenCV:
  `https://www.geeksforgeeks.org/python-opencv-cv2-imread-method/`

5. Camera Calibration and 3D Reconstruction (calib3d module)

- OpenCV Docs – Camera Calibration and 3D Reconstruction:
  `https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html`

- LearnOpenCV – Camera Calibration Using OpenCV:
  `https://learnopencv.com/camera-calibration-using-opencv/`

6. Object Detection (objdetect module)

- OpenCV Docs – Object Detection:
  `https://docs.opencv.org/4.x/d5/d54/group__objdetect.html`
- PyImageSearch – Face Detection with Haar Cascades:
  `https://pyimagesearch.com/2021/04/19/face-detection-with-haar-cascades-in-opencv/`

7. 2D Features Framework (feature2d module)

- OpenCV Docs – 2D Features Framework:
  `https://docs.opencv.org/4.x/db/d27/tutorial_py_table_of_contents_feature2d.html`
- LearnOpenCV – Feature Detection (SIFT, ORB, etc.):
  `https://learnopencv.com/feature-detection-using-opencv/`

8. Deep Neural Networks (dnn module)

- OpenCV Docs – Deep Neural Networks (dnn):
  `https://docs.opencv.org/4.x/d6/d0f/group__dnn.html`
- PyImageSearch – Deep Learning with OpenCV:
  `https://pyimagesearch.com/category/deep-learning/`

9. Graph API (gapi module)

- OpenCV Docs – Graph API (G-API):
  `https://docs.opencv.org/4.x/d0/d1e/gapi.html`

- OpenCV Blog – Introduction to G-API:
  https://opencv.org/opencv-g-api/

10. Other Tutorials (ml, photo, stitching, video)

- OpenCV Docs – Machine Learning Module:
  https://docs.opencv.org/4.x/d0/d61/tutorial_ml_intro.html
- OpenCV Docs – Photo Module:
  https://docs.opencv.org/4.x/d4/d1e/tutorial_photo.html
- OpenCV Docs – Image Stitching:
  https://docs.opencv.org/4.x/d8/d19/tutorial_stitcher.html
- OpenCV Docs – Video Analysis:
  https://docs.opencv.org/4.x/d7/de9/tutorial_video_input_psnr_ssim.html

11. Theory of Image Processing

- Gonzalez and Woods – Digital Image Processing (4th Edition).
- Khan Academy – Image Processing Fundamentals:
  https://www.khanacademy.org/computing/pixels-and-image-processing
- GeeksforGeeks – Convolution, Edge Detection, and Fourier Transform:
  https://www.geeksforgeeks.org/introduction-to-image-processing/

12. Miscellaneous Items

- Research on OpenCV applications in robotics, medical imaging, autonomous vehicles, and surveillance – IEEE Xplore, SpringerLink.
- OpenCV Forum – Differences in Window Handling on Linux vs Windows:
  https://forum.opencv.org/

- GitHub Repo (pjreddie) – OpenCV Exercises Reference:
  `https://github.com/pjreddie`

C. Key Learnings and Insights

1. Understanding OpenCV's Modular Architecture

One of the key insights from this week's study is that OpenCV is organized into highly specialized modules, each targeting a particular area of computer vision or image processing. For example, `core` handles fundamental matrix operations and data structures, `imgproc` deals with image transformations, while `dnn` enables integration of deep learning models. Recognizing this modular structure makes it easier to navigate the documentation, import only the necessary modules, and optimize application performance.

2. Integration of Theory and Practice in Image Processing

Practical experimentation with operations like filtering, thresholding, and geometric transformations reinforced the underlying theoretical concepts such as convolution, kernel design, and interpolation. For example, applying Gaussian blur in OpenCV directly connects to understanding how a Gaussian kernel smooths high-frequency components, which helps in noise reduction. This interplay between theory and practice deepens both algorithmic understanding and coding proficiency.

3. Cross-Platform Differences and Development Considerations

Investigating OpenCV window handling revealed that the GUI backend behaves differently on Linux and Windows. While Windows can handle multiple named windows in separate threads relatively smoothly, Linux often requires GUI operations to remain in the main thread to avoid segmentation faults or unresponsive windows. Understanding

these platform-specific constraints is essential for writing portable code, especially when building interactive image processing tools.

4. Camera Calibration and 3D Reconstruction Insights

Working through camera calibration steps highlighted the importance of accurate corner detection, intrinsic/extrinsic parameter estimation, and distortion correction. Errors in calibration propagate into 3D reconstruction, stereo matching, or pose estimation tasks, making precise calibration a prerequisite for robotics, AR, and autonomous navigation projects.

5. Feature Detection and Matching in Real-World Applications

Exploring SIFT, ORB, and other feature detectors revealed their trade-offs: SIFT provides robust, scale- and rotation-invariant keypoints but is computationally expensive, whereas ORB offers faster execution suitable for real-time systems but may be less robust to extreme transformations. This understanding is vital for choosing the right feature extractor in resource-constrained applications.

6. Deep Neural Network (dnn) Module Usage

The `dnn` module's ability to load models from TensorFlow, Caffe, or ONNX without a heavy deep learning framework is highly valuable for deployment. However, experiments showed that inference speed and memory usage can vary significantly based on the model's complexity and the hardware backend (CPU vs GPU). Selecting the correct backend/target pairing can dramatically improve performance.

7. Importance of Pre-Processing in Model Performance

Both classical image processing and deep learning models benefit greatly from correct pre-processing—such as resizing, color space conversion, and normalization. Consistent

pre-processing ensures better reproducibility of results and reduces unexpected errors when deploying models on real-world data.

8. Practical Challenges and Debugging Strategies

Challenges faced included:

- Handling different image color formats (BGR in OpenCV vs RGB in Matplotlib).
- Managing file paths and encodings when reading/writing images.
- Avoiding mismatches between OpenCV's `uint8` image format and floating-point computations in NumPy.

Debugging strategies involved visualizing intermediate outputs, checking image shapes and data types, and incrementally building pipelines to isolate problematic steps.

9. Application Potential Across Industries

Research revealed that OpenCV is a core tool in domains such as:

- **Robotics:** Real-time visual navigation and SLAM.
- **Healthcare:** Medical image segmentation and analysis.
- **Autonomous Vehicles:** Lane detection, object recognition, and depth estimation.
- **Security & Surveillance:** Face detection, motion tracking, and behavior analysis.

This versatility is a result of OpenCV's broad function set and integration capabilities with other libraries.

10. Interconnection Between Modules

A significant takeaway is that OpenCV modules rarely work in isolation. For instance, a face detection pipeline may use:

1. `imgproc` for pre-processing,
2. `objdetect` for detection,
3. `highgui` for display,
4. and optionally `dnn` for deep learning-based refinement.

Recognizing these interconnections allows for designing more efficient, modular, and maintainable computer vision applications.

D. Example Code Snippets

1. Opening and Displaying an Image

```
# Opening image file
import cv2 as cv
import sys
img = cv.imread(cv.samples.findFile("Starry_Night.jpg"))
if img is None:
    sys.exit("Could not read the image.")
cv.imshow("Display window", img)
k = cv.waitKey(0)

if k == ord("s"):
    cv.imwrite("Starry_Night.jpg", img)
cv.destroyAllWindows()
```

Explanation: This snippet demonstrates how to load an image from disk, display it in an OpenCV window, and optionally save it when the user presses the 's' key. The program ensures graceful termination if the image file is not found.

2. Capturing and Displaying Webcam Feed in Grayscale

```
# Opening video stream from webcam
import numpy as np
import cv2 as cv
cap = cv.VideoCapture(0)
if not cap.isOpened():
    print("Cannot open camera")
    exit()

while True:
    ret, frame = cap.read()
    if not ret:
        print("Can't receive frame (stream end?). Exiting ...")
        break
    gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    cv.imshow('frame', gray)
    if cv.waitKey(1) == ord('q'):
        break

cap.release()
cv.destroyAllWindows()
```

Explanation: This code connects to the system webcam, captures frames in real-time, converts them to grayscale, and displays them. The video stream terminates when 'q' is

pressed.

3. Saving Video from Webcam

```
# Video saving example
import numpy as np
import cv2 as cv

cap = cv.VideoCapture(0)

# Define the codec and create VideoWriter object
fourcc = cv.VideoWriter_fourcc(*'XVID')
out = cv.VideoWriter('output.avi', fourcc, 20.0, (640, 480))

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        print("Can't receive frame (stream end?). Exiting ...")
        break
    out.write(frame)
    cv.imshow('frame', frame)
    if cv.waitKey(1) == ord('q'):
        break

cap.release()
out.release()
cv.destroyAllWindows()
```

Explanation: This script captures video from the webcam and saves it to disk in AVI

format using the XVID codec. It also displays the live feed while recording.

4. Accessing Pixel Values

```
import numpy as np
import cv2 as cv
img = cv.imread('messi.jpg')
assert img is not None, "file could not be read, check with os.path.exists()"
px = img[100,100]
print(px)
```

Explanation: This snippet demonstrates how to access pixel values from an image in OpenCV. The pixel at coordinates (100, 100) is retrieved and printed as a BGR triplet.

Output: `[B, G, R] values printed in console`

[64 37 47]

5. Accessing Only the Blue Channel

```
# Accessing only blue pixel
blue = img[100,100,0]
print(blue)
```

Explanation: This code accesses only the blue component of the pixel at coordinates (100, 100) in the loaded image.

Output: `Blue intensity value printed in console`

D.3 Gaussian Blur

```
# Apply Gaussian Blur
blur = cv.GaussianBlur(image, (5, 5), cv.BORDER_DEFAULT)
```

```
cv.imshow('Blurred Image', blur)
cv.waitKey(0)
cv.destroyAllWindows()
```

Explanation: This snippet applies Gaussian smoothing to the image, which helps in reducing noise and detail. The `(5, 5)` parameter specifies the kernel size for the blur.

D.4 Edge Detection with Canny

```
# Perform Canny edge detection
edges = cv.Canny(image, 125, 175)

cv.imshow('Edges', edges)
cv.waitKey(0)
cv.destroyAllWindows()
```

Explanation: The `cv.Canny()` function detects edges in the image by finding areas with rapid intensity changes. The threshold values (125, 175) control the sensitivity of edge detection.

Output

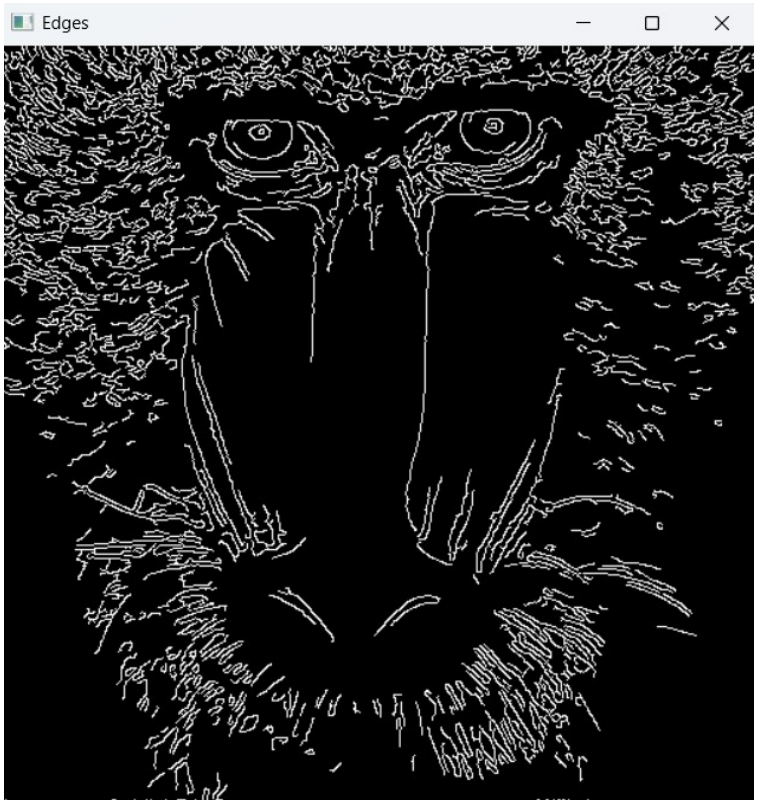

Figure 1: Canny edge detection

D.5 Drawing Shapes and Text

```
# Create a blank canvas
blank = np.zeros((500, 500, 3), dtype='uint8')

# Draw a rectangle
cv.rectangle(blank, (50, 50), (250, 250), (0, 255, 0), thickness=2)

# Draw a circle
cv.circle(blank, (300, 300), 50, (255, 0, 0), thickness=-1)
```

```
# Put text on image
cv.putText(blank, 'OpenCV Demo', (50, 400),
           cv.FONT_HERSHEY_SIMPLEX, 1.0, (255, 255, 255), thickness=2)

cv.imshow('Drawing', blank)
cv.waitKey(0)
cv.destroyAllWindows()
```

Explanation: This snippet shows how to create a blank canvas using NumPy and draw shapes such as rectangles and circles, as well as add text using `cv.putText()`. This is useful for annotations and overlays in image processing projects.

Output:

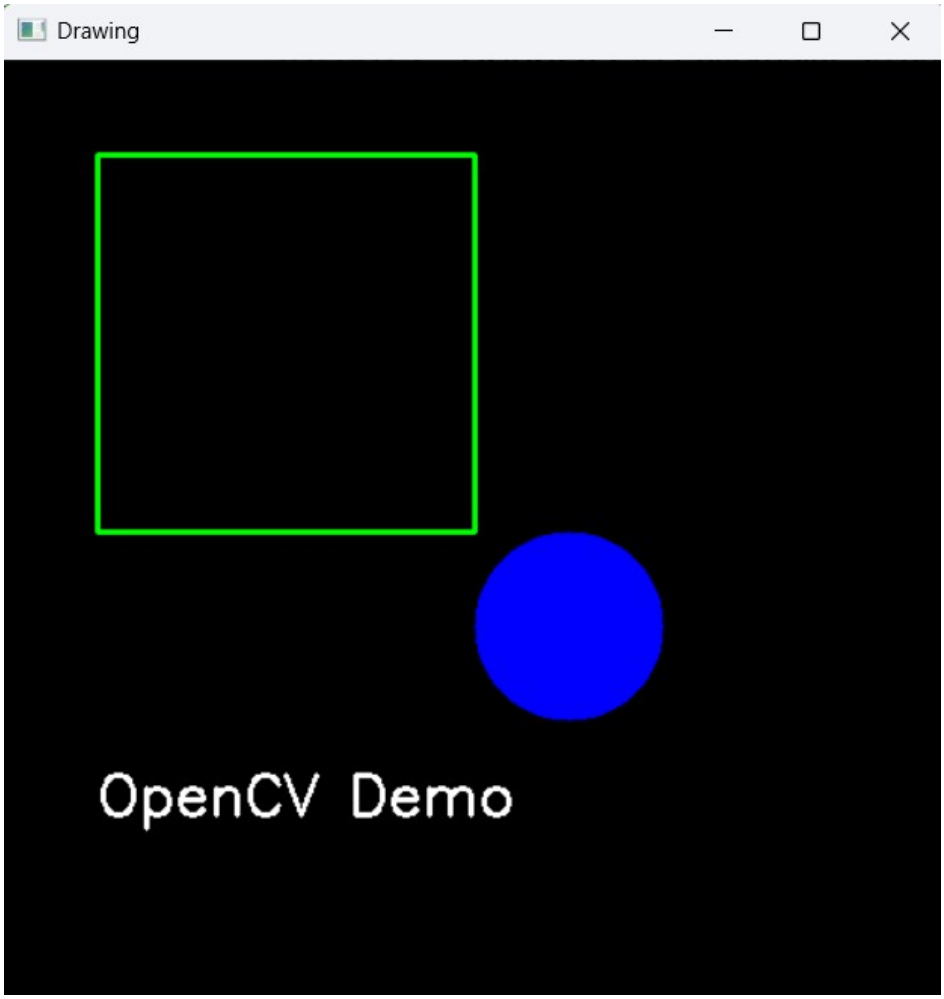

Figure 2: Drawing Shapes and Text

E. Conclusion

This week's exploration of OpenCV fundamentals has provided both theoretical insights and practical skills essential for computer vision development. Understanding the modular design of OpenCV has clarified how different components such as `core`, `imgproc`, `objdetect`, and `dnn` work together to form complete processing pipelines.

Hands-on practice with image reading/writing, video capture, geometric transformations, filtering, and feature detection reinforced the connection between underlying mathematical concepts and their practical implementation. Challenges such as cross-platform window handling, data type mismatches, and efficient pre-processing highlighted the importance of careful coding and debugging in real-world projects.

Beyond technical skills, the assignment emphasized OpenCV's versatility across industries — from robotics and autonomous vehicles to medical diagnostics and security systems. By combining theoretical study, experimentation, and research into advanced features like deep neural network integration, the work completed this week lays a solid foundation for more complex vision-based applications in the future.