**Practical - 4**

**Local Image Resolution Reduction Using Docker**

**Introduction to Docker**

Docker is an open-source platform that enables developers to build, deploy, and run applications in lightweight, portable containers. Unlike traditional virtual machines, Docker containers share the host system's kernel while maintaining isolated user spaces, making them highly efficient for application deployment. Containers package an application with all its dependencies, ensuring consistent execution across different computing environments.

**Project Overview**

This assignment demonstrates a practical implementation of Docker for batch image processing. We develop a containerized solution that reduces image resolution offline using Python's Pillow library. The approach highlights Docker's capabilities in creating reproducible, platform-independent environments for computational tasks.

**Technical Implementation**

**System Architecture**

The solution employs a three-layer architecture:

1. Host System: Provides Docker runtime environment

2. Container Layer: Isolated execution environment with Python and Pillow

3. Storage Interface: Bidirectional volume mapping between host and container

**Key Components**

1. Docker Containerization

- Utilizes lightweight Python-alpine base image

- Pre-installed Pillow library for image processing

- Configurable through Dockerfile directives

2. Image Processing Workflow

- Input directory scanning

- Resolution transformation algorithm

- Batch processing capability

3. Execution Pipeline

- Docker image construction

- Container instantiation with volume binding

- Output generation in host directory

**Implementation Process**

**Environment Configuration**

The solution requires:

- Docker Engine installation

- Proper directory structure for I/O operations

- Python environment within container

**Operational Workflow**

1. Docker image building via Dockerfile

2. Container execution with volume mounting

3. Automated batch processing of images

4. Output storage in designated host directory

**Advantages of Docker-Based Approach**

**Technical Benefits**

- Environment Consistency: Uniform execution across platforms

- Isolation: Process separation without VM overhead

- Resource Efficiency: Shared kernel architecture minimizes resource usage

**Practical Advantages**

- Portability: "Build once, run anywhere" capability

- Reproducibility: Version-controlled dependencies

- Scalability: Easily adaptable for large-scale processing

**Conclusion**

This project successfully demonstrates Docker's effectiveness in creating portable solutions for image processing tasks. The containerized approach offers significant improvements in deployment flexibility and environment management compared to traditional methods. Future enhancements could include:

- Dynamic resolution parameters

- Quality optimization controls

- Parallel processing implementation

The implementation serves as a practical example of how container technology can solve real-world computational challenges while maintaining efficiency and cross-platform compatibility.

**Code -**

from PIL import Image

import os

import sys

def reduce\_resolution(input\_path, output\_path, scale\_factor=0.5):

    try:

        with Image.open(input\_path) as img:

            # Calculate new dimensions

            width, height = img.size

            new\_width = int(width \* scale\_factor)

            new\_height = int(height \* scale\_factor)

            # Resize the image

            resized\_img = img.resize((new\_width, new\_height), Image.LANCZOS)

            # Save the resized image

            resized\_img.save(output\_path)

            print(f"Image saved to {output\_path}")

    except Exception as e:

        print(f"Error: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

    if len(sys.argv) < 3:

        print("Usage: python reduce\_resolution.py <input\_image> <output\_image> [scale\_factor]")

        sys.exit(1)

    input\_image = sys.argv[1]

    output\_image = sys.argv[2]

    scale\_factor = float(sys.argv[3]) if len(sys.argv) > 3 else 0.5

    reduce\_resolution(input\_image, output\_image, scale\_factor)

**DockerFile –**

# Use official Python image

FROM python:3.9-alpine

# Set working directory

WORKDIR /app

# Install dependencies

RUN pip install Pillow

# Copy the script

COPY reduce\_resolution.py .

# Command to run the script

ENTRYPOINT ["python", "reduce\_resolution.py"]





 