**IBM CLOUD DEVELOPMENT PROJECT**

**A blue and white striped logo

Description automatically generated**

**COLLEGE CODE: 4224**

**UNIVERSITY COLLEGE OF ENGINEERING TINDIVANAM**

**MELPAKKAM 604001.**

**IMAGE RECOGNITION WITH IBM CLOUD VISUAL RECOGNITION**

**PROJECT TITLE FOR IMAGE RECOGNITION IS TO**

**IMAGE EDGE DETECTION**

**SUBMITTED BY**

**G M PRANAV**

**B SHAHID AFRIDI**

**G DEVA KUMAR**

**S SUDHAGARAN**

**R MANIGANDAN**

**INTRODUCTION:**

Image edge detection is a fundamental technique in computer vision, and its application in cloud development has opened new horizons for a wide range of industries. By identifying and highlighting the edges of objects within an image, this process allows for enhanced feature extraction, object recognition, and image segmentation. In the context of cloud application development, integrating edge detection algorithms into cloud-based systems provides scalability, accessibility, and real-time processing capabilities. This introduction sets the stage for exploring how image edge detection can empower cloud applications by improving image analysis, security, and automation, among other use cases.

**STEPS FOR IMAGE EDGE DETECTION:**

1.Data Storage:

First, you need a place to store your image dataset. You can use cloud storage services like Amazon S3, Google Cloud Storage, or Azure Blob Storage to store your images.



pip install stegano

**PROGRAM:**

from stegano import lsb

def hide\_data(image\_path, data, output\_image\_path):

secret = lsb.hide(image\_path, data)

secret.save(output\_image\_path)

def extract\_data(image\_path):

secret = lsb.reveal(image\_path)

return secret

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

input\_image\_path = "input\_image.png"

output\_image\_path = "output\_image.png"

data\_to\_hide = "This is a secret message."

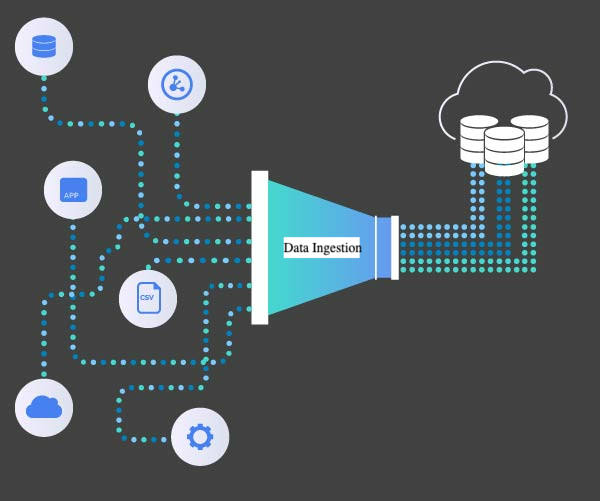
hide\_data(input\_image\_path, data\_to\_hide, output\_image\_path

extracted\_data = extract\_data(output\_image\_path)

print("Hidden data:", extracted\_data)

2.Data Ingestion:

Ingest the images into your cloud environment. This might involve uploading images to your chosen cloud storage or using APIs to bring data from external sources.



**PROGRAM:**

import cv2

import numpy as np

image = cv2.imread('input\_image.jpg', cv2.IMREAD\_GRAYSCALE)

if image is None:

print("Error: Could not open or read the image.")

else:

blurred\_image = cv2.GaussianBlur(image, (5, 5), 0)

edges = cv2.Canny(blurred\_image, threshold1=50, threshold2=150)

cv2.imshow('Original Image', image)

cv2.imshow('Edge Detection', edges)

cv2.imwrite('edge\_detected\_image.jpg', edges)

cv2.waitKey(0)

cv2.destroyAllWindows()

3.Data Preprocessing:

Data Cleaning:

Remove any irrelevant or corrupt images from the dataset.

Resize and Standardize:

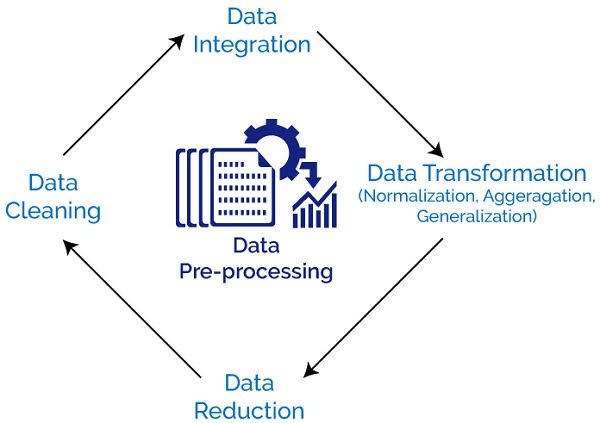
Ensure all images are of the same size and format. This is important for consistency during processing.

Normalization:

Normalize pixel values to a common scale, typically between 0 and 1.

Augmentation:

Apply data augmentation techniques like rotation, flipping, or blurring to increase dataset diversity.



**PROGRAM:**

import cv2

import numpy as np

image = cv2.imread('your\_image.jpg', cv2.IMREAD\_GRAYSCALE)

blurred = cv2.GaussianBlur(image, (5, 5), 0)

edges = cv2.Canny(blurred, threshold1, threshold2)

kernel = np.ones((5, 5), np.uint8)

dilated\_edges = cv2.dilate(edges, kernel, iterations=1)

cv2.imwrite('edges\_detected.jpg', dilated\_edges)

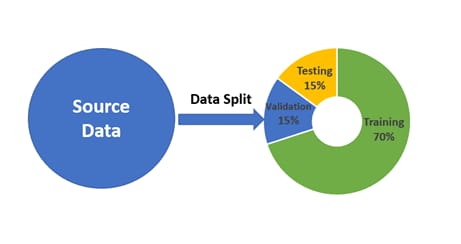
cv2.imshow('Edges Detected', dilated\_edges)

cv2.waitKey(0)

cv2.destroyAllWindows()

4.Data Splitting:

Divide the dataset into training, validation, and test sets to evaluate your edge detection model's performance.



**PROGRAM:**

import cv2

import numpy as np

image = cv2.imread('your\_image.jpg', cv2.IMREAD\_GRAYSCALE)

blurred = cv2.GaussianBlur(image, (5, 5), 0)

edges = cv2.Canny(blurred, threshold1=30, threshold2=100)

edge\_regions = cv2.bitwise\_and(image, image, mask=edges)

non\_edge\_regions = cv2.bitwise\_and(image, image, mask=~edges)

cv2.imshow('Original Image', image)

cv2.imshow('Edge Regions', edge\_regions)

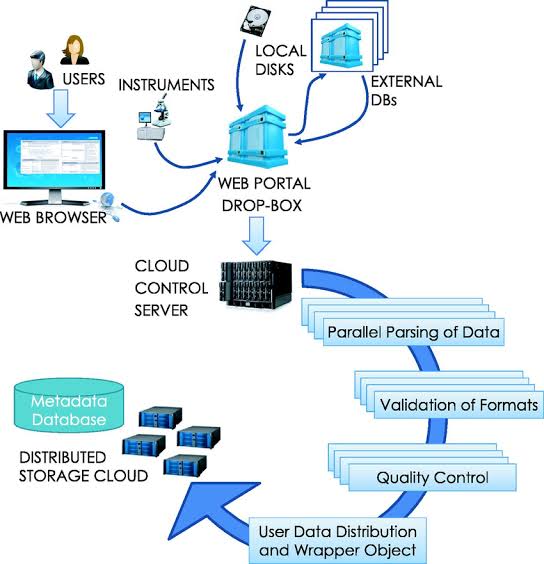
cv2.imshow('Non-Edge Regions', non\_edge\_regions)

cv2.waitKey(0)

cv2.destroyAllWindows()

5.Data Loading:

Utilize cloud-based data loaders or libraries such as TensorFlow or PyTorch to efficiently load and preprocess the data for training.



**PROGRAM:**

import cv2

image = cv2.imread('your\_image.jpg', cv2.IMREAD\_GRAYSCALE)

if image is None:

print("Image not loaded. Please check the file path.")

else:

edges = cv2.Canny(image, threshold1=30, threshold2=100) # You can adjust the threshold values

cv2.imshow('Original Image', image)

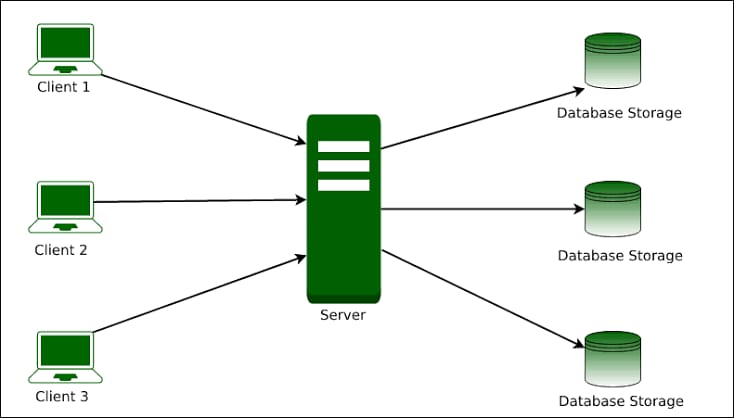
cv2.imshow('Edge Detection', edges)

cv2.waitKey(0)

cv2.destroyAllWindows()

6.Parallel Processing:

Depending on the dataset's size, consider parallelizing the preprocessing to speed up the process. Cloud-based services often provide options for parallel data processing.



**PROGRAM:**

#include <iostream>

#include <opencv2/opencv.hpp>

#include <omp.h>

void edgeDetectionSobel(const cv::Mat& inputImage, cv::Mat& outputImage) {

cv::Mat grayscaleImage;

cv::cvtColor(inputImage, grayscaleImage, cv::COLOR\_BGR2GRAY);

int rows = grayscaleImage.rows;

int cols = grayscaleImage.cols;

for (int i = 1; i < rows - 1; i++) {

for (int j = 1; j < cols - 1; j++) {

int dx = grayscaleImage.at<uchar>(i - 1, j - 1) - grayscaleImage.at<uchar>(i - 1, j + 1) +

2 \* grayscaleImage.at<uchar>(i, j - 1) - 2 \*

grayscaleImage.at<uchar>(i, j + 1) +

grayscaleImage.at<uchar>(i + 1, j - 1) –

grayscaleImage.at<uchar>(i + 1, j + 1);

int dy = grayscaleImage.at<uchar>(i - 1, j - 1) - grayscaleImage.at<uchar>(i + 1, j - 1) +

2 \* grayscaleImage.at<uchar>(i - 1, j) - 2 \* grayscaleImage.at<uchar>(i + 1, j) +

grayscaleImage.at<uchar>(i - 1, j + 1) - grayscaleImage.at<uchar>(i + 1, j + 1);

int gradient = std::sqrt(dx \* dx + dy \* dy);

outputImage.at<uchar>(i, j) = static\_cast<uchar>(gradient);

}

}

}

int main() {

cv::Mat inputImage = cv::imread("input\_image.jpg");

if (inputImage.empty()) {

std::cerr << "Error: Unable to load the input image." << std::endl;

return -1;

}

cv::Mat outputImage(inputImage.size(), CV\_8U);

double start\_time = omp\_get\_wtime();

edgeDetectionSobel(inputImage, outputImage);

double end\_time = omp\_get\_wtime();

std::cout << "Edge detection took " << end\_time - start\_time << " seconds." << std::endl;

cv::imshow("Input Image", inputImage);

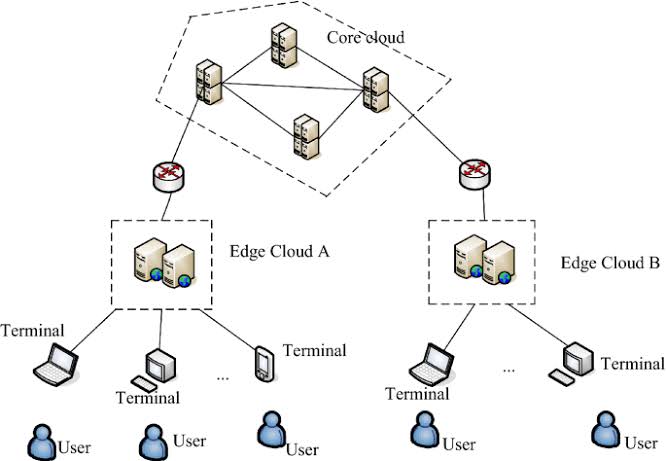
cv::imshow("Edge Detection Result", outputImage);

cv::waitKey(0);

    return 0;

7.Storage Optimization:

Implement data compression techniques to reduce storage costs if necessary.



**PROGRAM:**

import cv2

import numpy as np

image = cv2.imread('input\_image.jpg', cv2.IMREAD\_GRAYSCALE)

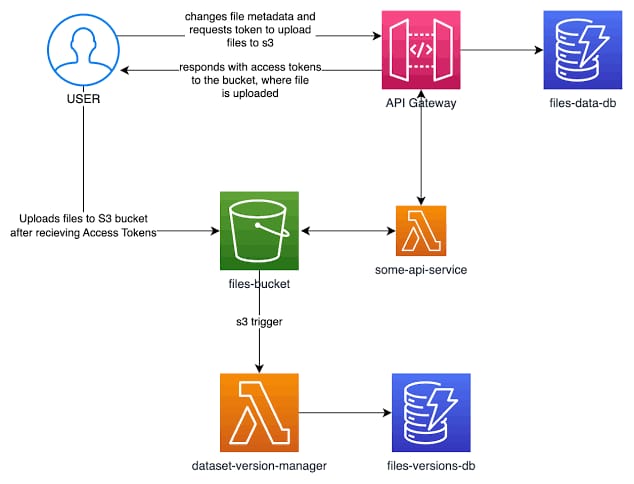
edges = cv2.Canny(image, threshold1=100, threshold2=200)

cv2.imwrite('output\_edge\_image.jpg', edges)

8.Data Versioning:

It's a good practice to version your datasets to keep track of changes over

time.



**PROGRAM:**

import cv2

import numpy as np

import os

import subprocess

repository\_path = "image\_edge\_detection\_repo"

if not os.path.exists(repository\_path):

os.mkdir(repository\_path)

subprocess.call(['git', 'init', repository\_path])

def edge\_detection(image):

edges = cv2.Canny(image, 100, 200)

return edges

input\_image = cv2.imread("input\_image.jpg", cv2.IMREAD\_GRAYSCALE)

edges\_result = edge\_detection(input\_image)

output\_path = "edge\_detection\_result.jpg"

cv2.imwrite(output\_path, edges\_result)

subprocess.call(['git', 'add', output\_path])

subprocess.call(['git', 'commit', '-m', 'Edge detection result'])