IBM CLOUD DEVELOPMENT PROJECT

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

K. KAVITHA (T.L)

R.RAMYA

V.SRISHA

INTRODUCTION:

Edge detection is a technique used in image processing and computer vision to identify the boundaries of objects within images. Specifically, it aims to pinpoint places in an image where there are abrupt changes in brightness or color. This is often one of the first steps in processing an image to extract meaningful information from it.The idea is that by identifying the edges in an image, you can determine the general shapes present, which can then be used for tasks like object detection, segmentation, and feature extraction.There are several algorithms and methods to perform edge detection, with these methods has its strengths and weaknesses, and the best one to use often depends on the specific application.

STEPS FOR IMAGE EDGE DETECTION:

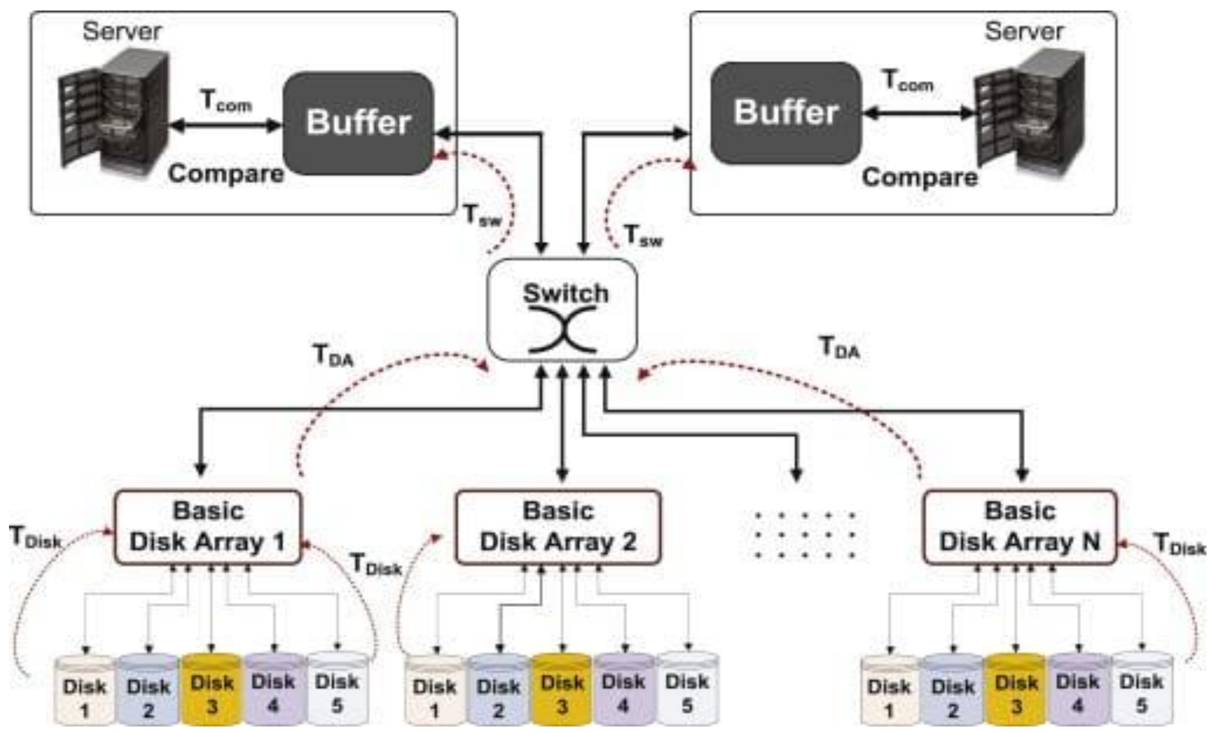
1.IMAGE DETECTION DATA STORAGE:

The method or system used to save, manage, and retrieve data resulting from or used in the process of image detection. This can include raw image data, processed image data, feature descriptors, machine learning models, metadata about detected objects or features, and any annotations or labels associated with the images.

THE SPECIFICS OF THIS STORAGE CAN VARY:

For large-scale image datasets, the data might be stored in distributed file systems or databases.In machine learning, the models trained for image detection (like Convolutional Neural Networks) would require storage for their weights, biases, and structure. This could be in local files or cloud storage.Annotations, which might be made by humans or automated processes to label parts of images, would need to be stored alongside the images or in associated databases.

DATA STORAGE ARCHITECTURE:



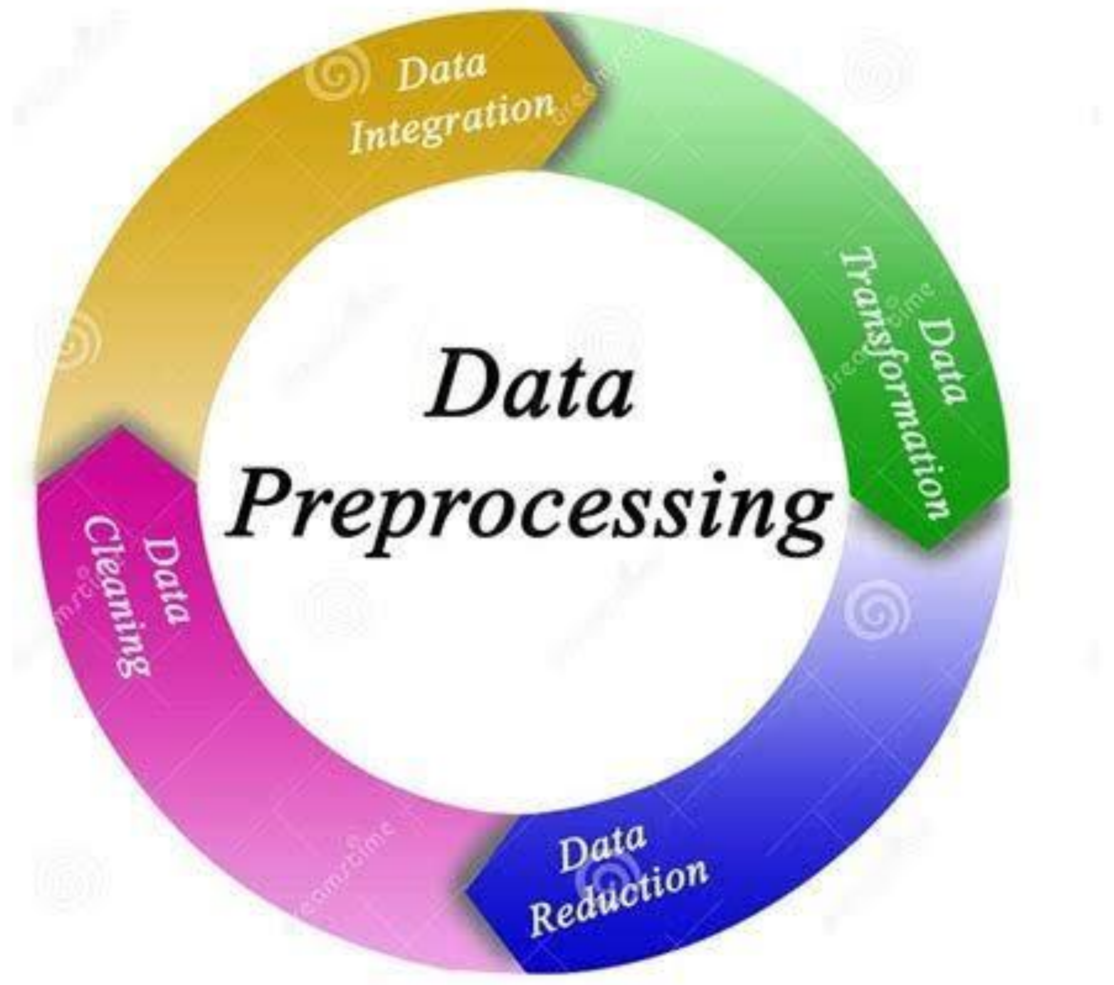
PROGRAM:

```
  import cv2

import json

def capture_image():

  cap = cv2.VideoCapture(0)

 if not cap.isOpened():

 print("Could not open webcam")

 exit()

 ret, frame = cap.read()

 cap.releas

 print("Could not read frame")

 exit()

def detect_faces(image):

face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_defau

  gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  faces = face_cascade.detectMultiScale(gray, 1.1, 4)

  return faces

 def save_to_file(data, filename):

 with open(filename, 'w') as file:

  json.dump(data, file)

 if _name_ == "_main_":

  img = capture_image()

 faces = detect_faces(img)

  face_data = [{"x": x, "y": y, "w": w, "h": h} for (x, y, w, h) in faces]

  save_to_file(face_data, "detected_faces.json")

  print("Faces detected and saved to detected_faces.json")
```

2.DATA PREPROCESSING:

Data preprocessing can refer to manipulation or dropping of data before it is used in order to ensure or enhance performance,[1] and is an important step in the data mining process. The phrase "garbage in, garbage out" is particularly applicable to data mining and machine learning projects. Data collection methods are often loosely controlled, resulting in out-of-range values, impossible data combinations, and missing values, amongst other issues.

Analyzing data that has not been carefully screened for such problems can produce misleading results. Thus, representation and quality of data is necessary before running any analysis.[2] Often, data preprocessing is the most important phase of a machine learning project, especially in computational biology.[3] If there is a high proportion of irrelevant and redundant information present or noisy and unreliable data, then knowledge discovery during the training phase may be more difficult. Data preparation and filtering steps can take a considerable amount of processing time.



3.DATA SPLITTING:

Splitting image data into train, validation, and test sets is a crucial step in machine learning model development. It helps to prevent over-fitting, evaluate model performance, and ensure that the model generalizes well to new, unseen data.

TRAINING SET:

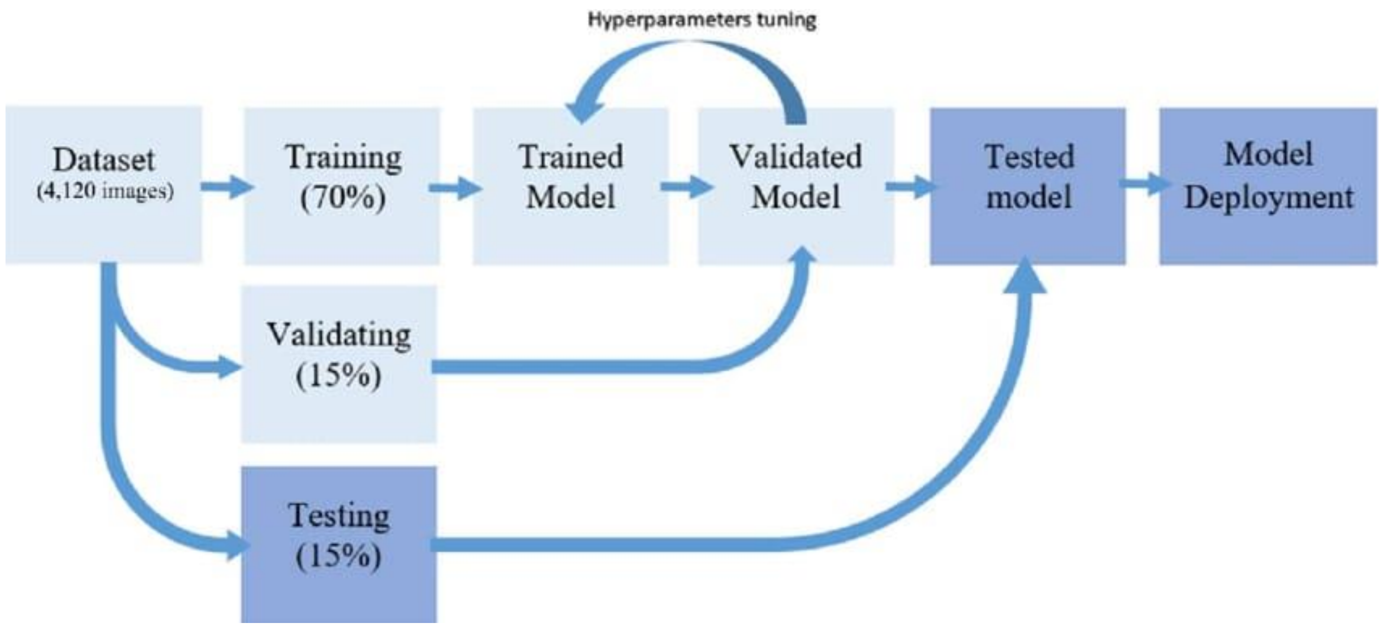
The largest portion of your data.Used to train the model.This is where the model learns the patterns in your data.

VALIDATION SET:

Used to fine-tune model parameters and prevent overfitting.It helps to get feedback on the model's performance during training.

TEST SET:

Used to evaluate the model's final performance after training.Should not be used in the training process at all.Gives an indication of how well the model will perform on unseen, real-world data.

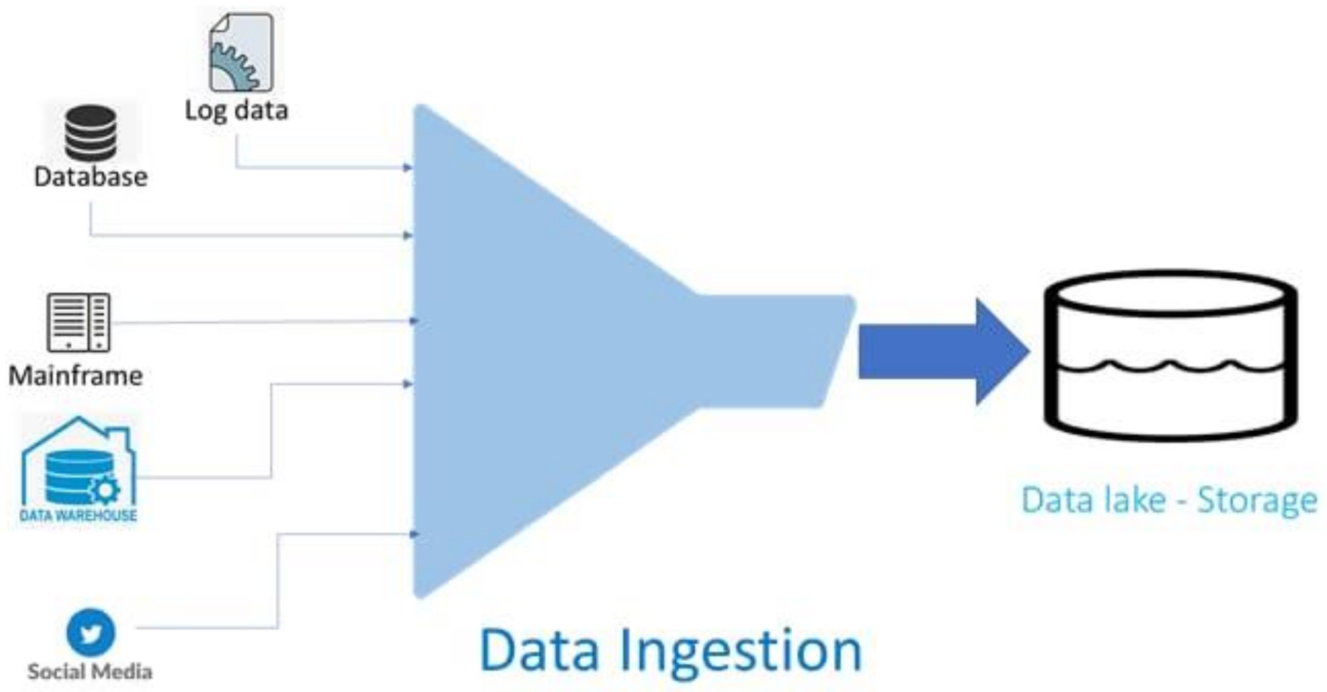


PROGRAM:

```
X= [1], [2], [3], [4], [5],  [6], [7], [8], [9], [10]

y = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("Training data:", X_train)

print("Testing data:", X_test)
```

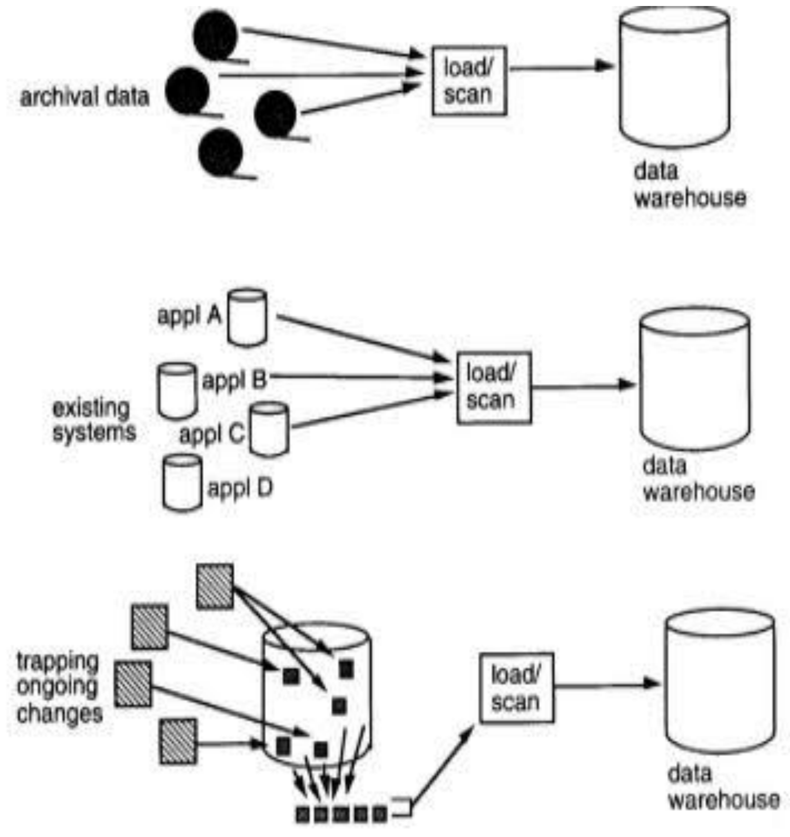
4.DATA INGESTION:

Data ingestion is the process of importing large, assorted data files from multiple sources into a single, cloud-based storage medium—a data warehouse, data mart or database—where it can be accessed and analyzed.



5.DATA LOADING:

Data loading is the process of copying and loading data or data sets from a source file, folder or application to a database or similar application. It is usually implemented by copying digital data from a source and



PROGRAM:

```
 import csv

def load_data(filename):

data = []

with open(filename, 'r') as file:

reader = csv.reader(file)

data.append(row)

return data

if _name_ == "_main_":

filename = "sample.csv"

loaded_data = load_data(filename)

print(row)
```

CONCLUSION:

Image detection, a subset of computer vision, refers to the capability of software and machines to identify objects, features, or activities in images. It can be used for various applications, such as facial recognition, autonomous vehicles, and medical imaging.