**VARUVAN VADIVELAN INSTITUTE**

**OF TECHNOLOGY**

**NAAN MUDHALVAN: IBM**

**TECHNOLOGY: CLOUD APPLICATION**

**DEVELOPMENT-PHASE 3**

**PROJECT TITLE: IMAGE RECOGNITION WITH IBM CLOUD VISUAL RECOGNITION**

**DEVELOPMENT PART:**

Image recognition systems typically utilize advanced algorithms, machine learning techniques, and deep neural networks to analyse and classify visual data. These systems are trained on large datasets of labeled images, allowing them to learn the features and patterns associated with different classes or categories of objects.

**IMAGE CLASSIFICATION:**

Image classification in cloud computing refers to the process of

categorizing or labeling images into predefined classes or

categories using cloud-based resources, particularly machine

learning or deep learning models and services.

**PROGRAM:**

import tensorflow as tf

from tensorflow.keras import datasets, layers, models

import matplotlib.pyplot as plt

(train\_images, train\_labels), (test\_images, test\_labels) = datasets.cifar10.load\_data()

train\_images, test\_images = train\_images / 255.0, test\_images / 255.0

model = models.Sequential([

    layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)),

    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.Flatten(),

    layers.Dense(64, activation='relu'),s

    layers.Dense(10)

])

model.compile(optimizer='adam',

              loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

              metrics=['accuracy'])

history = model.fit(train\_images, train\_labels, epochs=10,

                    validation\_data=(test\_images, test\_labels))

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels, verbose=2)

print(f'Test accuracy: {test\_acc}')

**OUTPUT:**

Epoch 1/10

1563/1563 [==============================] - 70s 45ms/step - loss: 1.4985 - accuracy: 0.4542 - val\_loss: 1.2327 - val\_accuracy: 0.5595

...

Epoch 10/10

1563/1563 [==============================] - 70s 45ms/step - loss: 0.4567 - accuracy: 0.8341 - val\_loss: 0.9043 - val\_accuracy: 0.6893

313/313 - 4s - loss: 0.9043 - accuracy: 0.6893

Test accuracy: 0.6893000001907349

**IMAGE RECOGNITION:**

Image recognition in cloud computing refers to the process of

identifying and categorizing objects, patterns, or entities within

digital images using cloud-based resources and machine

learning or deep learning techniques.

**PROGRAM:**

import matplotlib.pyplot as plt

import numpy as np

import os

import PIL

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from tensorflow.keras.models import Sequential

# importing flower dataset

import pathlib

dataset\_url = "https://storage.googleapis.com/download.tensorflow.org/example\_images/flower\_photos.tgz"

data\_dir = tf.keras.utils.get\_file('flower\_photos', origin=dataset\_url, untar=True)

data\_dir = pathlib.Path(data\_dir)

image\_count = len(list(data\_dir.glob('\*/\*.jpg')))

print(image\_count)

roses = list(data\_dir.glob('roses/\*'))

PIL.Image.open(str(roses[0]))

**OUTPUT:**



**OBJECT DETECTION:**

Object detection in cloud computing refers to the

process of identifying and locating specific objects

within images or video streams using cloud-based resources and machine learning or deep learning

algorithms.

**PROGRAM:**

import cv2

net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")

classes = []

with open("coco.names", "r") as f:

    classes = [line.strip() for line in f.readlines()]

img = cv2.imread("example\_image.jpg")

height, width, \_ = img.shape

blob = cv2.dnn.blobFromImage(img, 1/255.0, (416, 416), swapRB=True, crop=False)

net.setInput(blob)

output\_layers\_names = net.getUnconnectedOutLayersNames()

layer\_outputs = net.forward(output\_layers\_names)

boxes = []

confidences = []

class\_ids = []

for output in layer\_outputs:

    for detection in output:

        scores = detection[5:]

        class\_id = np.argmax(scores)

        confidence = scores[class\_id]

        if confidence > 0.5:  # Consider only detections with confidence more than 50%

            center\_x = int(detection[0] \* width)

            center\_y = int(detection[1] \* height)

            w = int(detection[2] \* width)

            h = int(detection[3] \* height)

            x = int(center\_x - w / 2)

            y = int(center\_y - h / 2)

            boxes.append([x, y, w, h])

            confidences.append(float(confidence))

            class\_ids.append(class\_id)

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

colors = np.random.uniform(0, 255, size=(len(classes), 3))

for i in indexes:

    i = i[0]

    box = boxes[i]

    x, y, w, h = box

    label = str(classes[class\_ids[i]])

    confidence = confidences[i]

    color = colors[class\_ids[i]]

    cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)

    cv2.putText(img, f"{label} {confidence:.2f}", (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, color, 2)

cv2.imshow("Image", img)

cv2.waitKey(0)

cv2.destroyAllWindows()

**RUN CAMERA WITH CLOUD:**

To run a camera with cloud integration using coding, you typically need to use appropriate software development kits (SDKs) or application programming interfaces (APIs) provided by the cloud service and the camera manufacturer. Since camera and cloud technologies vary, here's a basic example using Python for integrating a camera with the IBM Cloud Object Storage:

import cv2

import ibm\_boto3

import ibm\_botocore

# Connect to IBM Cloud Object Storage

cos = ibm\_boto3.resource("s3",

    ibm\_api\_key\_id='<your\_ibm\_api\_key>',

    ibm\_service\_instance\_id='<your\_ibm\_service\_instance\_id>',

    ibm\_auth\_endpoint="https://iam.cloud.ibm.com/identity/token",

    config=ibm\_botocore.client.Config(signature\_version="oauth"),

    endpoint\_url='<your\_ibm\_cos\_endpoint>'

)

# Access the camera

camera = cv2.VideoCapture(0)  # Replace 0 with the appropriate camera index if needed

while True:

    # Capture frame-by-frame

    ret, frame = camera.read()

    # Display the resulting frame

    cv2.imshow('frame', frame)

    # Save the frame to IBM Cloud Object Storage

    bucket\_name = '<your\_bucket\_name>'

    file\_name = '<your\_file\_name>.jpg'

    cv2.imwrite(file\_name, frame)

    with open(file\_name, 'rb') as data:

        cos.Bucket(bucket\_name).put\_object(Key=file\_name, Body=data)

    if cv2.waitKey(1) & 0xFF == ord('q'):

        break

# When everything done, release the capture

camera.release()

cv2.destroyAllWindows()

**GSM GATEWAY:**

A GSM gateway is a device that allows users to route calls between the public switched telephone network (PSTN) and mobile networks. It essentially acts as a bridge between telecommunication networks that operate on different protocols, such as traditional landline networks (PSTN) and mobile networks.

**USER USING PYTHON VISUAL RECOGNITION:**

with location To use the Watson Visual Recognition service with location services in the IBM Cloud using Python, you'll need to install the required Python SDK and configure the Watson service. Here's a simplified example that demonstrates how to use Watson Visual Recognition services:

**PROGRAM:**

from ibm\_watson import VisualRecognitionV3

from ibm\_cloud\_sdk\_core.authenticators import IAMAuthenticator

# Authenticate the Watson service

authenticator = IAMAuthenticator('your\_api\_key')

visual\_recognition = VisualRecognitionV3(

    version='2018-03-19',

    authenticator=authenticator

)

# Set the URL of the image you want to analyze

url = 'URL\_of\_your\_image'

# Specify the location coordinates for the image

latitude = 'your\_latitude'

longitude = 'your\_longitude'

# Analyze the image using Watson Visual Recognition with location

response = visual\_recognition.classify(

    url=url,

    classifier\_ids=['default'],

    location=[latitude, longitude]

).get\_result()

# Print the result

print(response)

Make sure to replace 'your\_api\_key', 'URL\_of\_your\_image', 'your\_latitude', and 'your\_longitude' with your actual values. Also, ensure you have installed the necessary Python packages. You can install the required package using pip:

pip install ibm-watson

This script sends a request to the Watson Visual Recognition service to classify an image from a specified URL. The 'location' parameter specifies the geographical coordinates associated with the image. The script then prints the response received from the service.

Ensure that you have the necessary permissions and valid credentials to access the Watson Visual Recognition service and that the location services are properly configured in the IBM Cloud.

**OUTPUT:**

{

  "images": [

    {

      "classifiers": [

        {

          "classifier\_id": "default",

          "name": "default",

          "classes": [

            {

              "class": "animal",

              "score": 0.85

            },

            {

              "class": "dog",

              "score": 0.75

            },

            {

              "class": "mammal",

              "score": 0.9

            }

          ]

        }

      ],

      "image": "URL\_of\_your\_image"

    }

  ],

  "images\_processed": 1

}

**FINAL PROGRAM:**

Import json

From ibm\_watson import VisualRecognitionV3

From ibm\_cloud\_sdk\_core.authenticators import

IAMAuthenticator

From PIL import Image

Import requests

# Replace with your IBM Cloud API key and endpoint

Api\_key = ‘YOUR\_API\_KEY’

Service\_endpoint = ‘YOUR\_SERVICE\_ENDPOINT’

# Initialize the Visual Recognition service

Authenticator = IAMAuthenticator(api\_key)

Visual\_recognition = VisualRecognitionV3(

Version=’2018-03-19’,

Authenticator=authenticator

)

Visual\_recognition.set\_service\_url(service\_endpoint)

# URL of the image you want to classify

Image\_url = ‘URL\_TO\_IMAGE’

# Use the SDK to classify the image

Classes =

visual\_recognition.classify(url=image\_url).get\_result()

# Extract and print the results

If ‘images’ in classes and len(classes[‘images’]) > 0:

For cls in classes[‘images’][0][‘classifiers’][0][‘classes’]:

Print(f”Class: {cls[‘class’]}”)

Print(f”Score: {cls[‘score’]}”)

Else:

Print(“No classes found in the image.”)

# You can also use local image files by providing

‘image\_path’ instead of ‘url’

**OUTPUT:**

Class: cat

Score: 0.897

Class: animal

Score: 0.867

Class: pet

Score: 0.856

Class: mammal

Score: 0.754

Class: dog

Score: 0.678