For A MAJOR PROJECT REPORT ON

CONTENT BASED IMAGE RETRIEVAL USING CONVOLUTAIONAL NEURAL NETWORKS

Submitted in Partial Fulfilment of the requirements to
Major Project (CA-261)

II MCA - II Semester
Submitted

By

Tati Siva Sankar

(Y23CA053)

April-2025

R.V.R & J.C COLLEGE OF ENGINEERING (Autonomous)
(NAAC – "A+" Grade)
(Affiliated to Acharya Nagarjuna University, GUNTUR)
(Approved by AICTE & Accredited by NBA)
Chandramoulipuram: Chowdavaram
GUNTUR- 522019

CONTENT BASED IMAGE RETRIEVAL USING CONVOLUTAIONAL NEURAL NETWORKS

A Major Project Report Submitted
In Partial fulfillment of the Requirements for the Degree of

MASTER OF COMPUTER APPLICATIONS

By

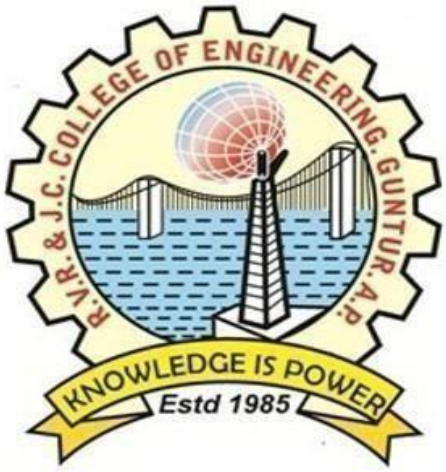
Tati Siva Sankar

(Y23CA053)

Under the guidance of

Dr. K. KARTEEKA PAVAN

Professor and Head of the Department of Computer Applications



April-2025

R.V.R & J.C COLLEGE OF ENGINEERING (Autonomous)

(NAAC – "A+" Grade)

(Affiliated to Acharya Nagarjuna University, GUNTUR)

(Approved by AICTE & Accredited by NBA)

Chandramoulipuram: Chowdavaram

GUNTUR- 522019

R.V.R & J.C COLLEGE OF ENGINEERING
(AUTONOMOUS)

DEPARTMENT OF COMPUTER APPLICATIONS

BONAFIDE CERTIFICATE

This is to certify that the major project work entitled "**Content Based Image Retrieval Using Convolutional Neural Networks**" has been carried out by **Tati Siva Sankar (Y23CA053)** under my guidance in partial fulfillment of the requirements of **CA261–MAJOR PROJECT WORK** for the completion of the course, **MASTER OF COMPUTER APPLICATIONS**, during the academic year 2024-2025.

Dr.K.Karteeka Pavan
Prof., & HoD

CERTIFICATE OF COMPLETION

To,

The Head Of The Department

Master Of Computer Applications

RVR&JC College Of Engineering

Chowdavaram, Guntur District, Andhra Pradesh 522019

CERTIFICATE OF COMPLETION

This is to certify that **T. Siva Sankar** studying in R V R & J C College of Engineering, GUNTUR, A.P with University Enrolment No **Y22CA053** successfully completed his project work titled **"Content Based Image Retrieval using CNN"**

He has done the project during the period from **09-01-2025 to 09-05-2025** under the guidance and supervision of **Mr. V Narayana Pasupuleti** , CEGON TECHNOLOGIES, VIJAYAWADA.

He has completed the assigned project well within the time frame and is sincere, hardworking and his conduct during his project is commendable.

We wish all the best to his future endeavors.

MANAGING DIRECTOR

PSP SHARMA

ACKNOWLEDGEMENT

The successful completion of any task would be incomplete without a proper suggestions, guidance and environment. Combination of these three factors acts like backbone to our Major Project "**Content Based Image Retrieval Using Convolutional Neural Networks**".

I am very glad to express our special thanks to **Dr. K. Karteeka Pavan**, Professor & HoD of MCA and our Major Project Guide, who has inspired us to select this topic, and also for her valuable advices in preparing this Project work.

I am very glad to express my special thanks to **Smt M. Chaitanya,** Asst.Prof**,** major project in-charge who was inspired me to select this topic and also for her valuable advices in preparing this thesis.

I am very much thankful to **Dr. Kolla Srinivas,** Principal of **RVR&JC COLLEGE OF ENGINEERING,** Guntur for having allowed to delivering this major project.

Finally, I submit my reserves thanks to Lab staff in **Department of Computer Applications** and to all my friends for their cooperation during the preparation and developing the project.

Tati Siva Sankar(Y23CA053)

CONTENT BASED IMAGE RETRIEVAL USING CONVOLUTAIONAL NEURAL NETWORKS

Tati Siva Sankar(Y23CA053)

ABSTRACT

This project explores a content-based image retrieval (CBIR) system designed to enhance the accuracy and efficiency of finding visually similar images. The COREL1000 dataset, consisting of diverse image categories, is utilized for training various state-of-the-art deep learning algorithms, including VGG16, MobileNet, and ResNet. These algorithms are employed for feature extraction, leveraging their advanced architectures to capture intricate image details and patterns. Once the images are retrieved using these deep learning models, a COSINE similarity technique is applied to compare the extracted features, enabling the system to identify and rank the top 10 most similar images. This hybrid approach combines the strengths of deep learning for robust feature extraction and COSINE similarity for precise similarity measurement. The proposed system demonstrates the potential of integrating deep learning and similarity metrics to improve the effectiveness of content-based image retrieval, making it suitable for applications in fields like multimedia search, digital libraries, and image-based recommendation systems.

CONTENTS

1.INTRODUCTION

1.1 Introduction

With the rapid growth of digital content, efficiently retrieving visually similar images from large databases has become a critical challenge. Traditional text-based image retrieval methods often fail to capture the true visual semantics of images, leading to inaccurate results. To address this limitation, Content-Based Image Retrieval (CBIR) systems have emerged, leveraging image features such as color, texture, and shape for more precise retrieval.

This project explores a CBIR system that enhances the accuracy and efficiency of image retrieval by integrating state-of-the-art deep learning algorithms. The system is trained on the COREL1000 dataset, a well-known benchmark containing diverse image categories. Deep learning models such as VGG16, MobileNet, and ResNet are employed for feature extraction, capturing intricate patterns and details from images.

Once the feature representations are extracted, a COSINE similarity technique is applied to measure the similarity between images. By ranking and retrieving the top 10 most similar images, the system ensures highly relevant search results. This hybrid approach—combining deep learning for feature extraction and COSINE similarity for comparison—demonstrates significant improvements in retrieval performance.

The proposed CBIR system has broad applications in various domains, including multimedia search, digital libraries, and image-based recommendation systems. By integrating deep learning with similarity metrics, this project showcases a powerful and efficient solution for image retrieval, paving the way for advanced applications in visual data processing.

1.2 Proposed System

In this project used COREL1000 images dataset to train various deep learning algorithms such as VGG16, MobileNet and ResNet to predict content based images retrieval. After retrieving images using deep learning algorithms then we are applying COSINE similarity to fetch top 10 similar images.

1.2.1 Advantages of the Proposed System

Improved Accuracy – CNN-based models like VGG16, MobileNet, and ResNet effectively extract deep features, leading to more accurate image retrieval compared to traditional methods.

Automated Feature Extraction – Unlike traditional CBIR techniques that rely on handcrafted features, deep learning models automatically learn and extract relevant image features.

High Performance – Using pre-trained deep learning models enhances retrieval speed and accuracy, reducing the need for manual feature engineering.

1.3 System Requirement Specification

1.3.1 SOFTWARE REQUIREMENTS

- Software : Anaconda
- Primary Language : Python
- Frontend Framework : Flask
- Back-end Framework : Jupyter Notebook
- Database : Sqlite3
- Front-End Technologies : HTML,CSS,JavaScript and Bootstrap4

1.3.2 HARDWARE REQUIREMENTS

- Operating System : Windows Only
- Processor : i5 and above
- Ram : 8 GB and above
- Hard Disk : 25 GB in local drive

2. SYSTEM ANALYSIS

2.1 Problem Statement

Finding visually similar images efficiently and accurately is a challenging task in content-based image retrieval (CBIR) systems. Traditional methods often struggle with capturing intricate image details and patterns, leading to suboptimal retrieval performance. There is a need for a robust approach that can extract meaningful image features and effectively compare them to improve similarity measurement.

2.2 Existing System

Searchable encryption (SE) enables the clients to store the encrypted data at the cloud, meanwhile supports data search .However, many of the existing SE schemes are designed for text documents proposed the first privacy-preserving CBIR scheme over the encrypted image database. The scheme utilized the set of visual words to represent images. The similarity between images was measured by Jaccard distance between the sets of visual words. The min-hash algorithm and order-preserving encryption were employed to protect the visual words.

2.2.1 Disadvantages of the Existing System

An existing methodology doesn’t implement Privacy-preserving CBIR protocol Method.The system not implemented Bag-of-Word model Technique.

2.3 UML Diagrams

2.3.1 Use Case Diagram

A use case diagram visually represents the interactions between actors and the system's use cases in the Detection of abnormal activities in online exams using pre-trained neural networks project. Actors include students and educators, while use cases encompass monitoring behaviours, reporting results, and managing exams. Associations illustrate how actors engage with these use cases, helping to clarify system functionality and user requirements.

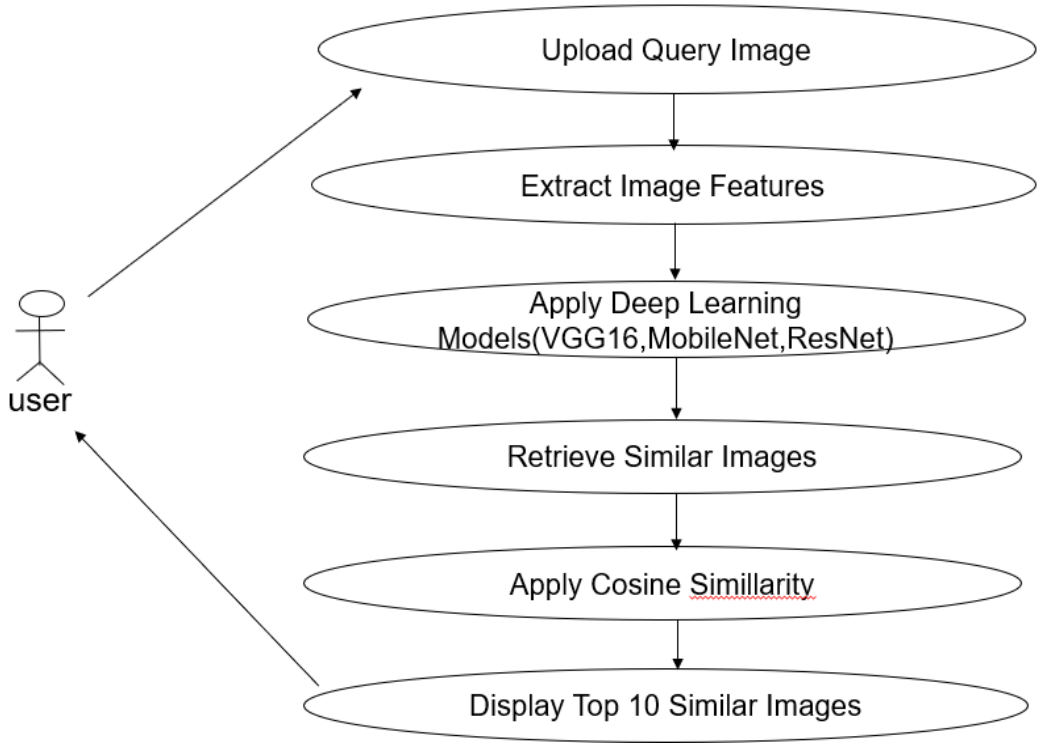


Fig 2.3.1 Use Case Diagram

2.3.2 Class Diagram

A class diagram defines the structure of the detection of abnormal activities in online exams using pre-trained neural networks project by illustrating classes, attributes, and operations. Classes represent system components, such as User and Exam, with attributes detailing their characteristics. Operations define methods for handling behaviours, while aggregation shows relationships between classes, aiding in understanding system architecture.

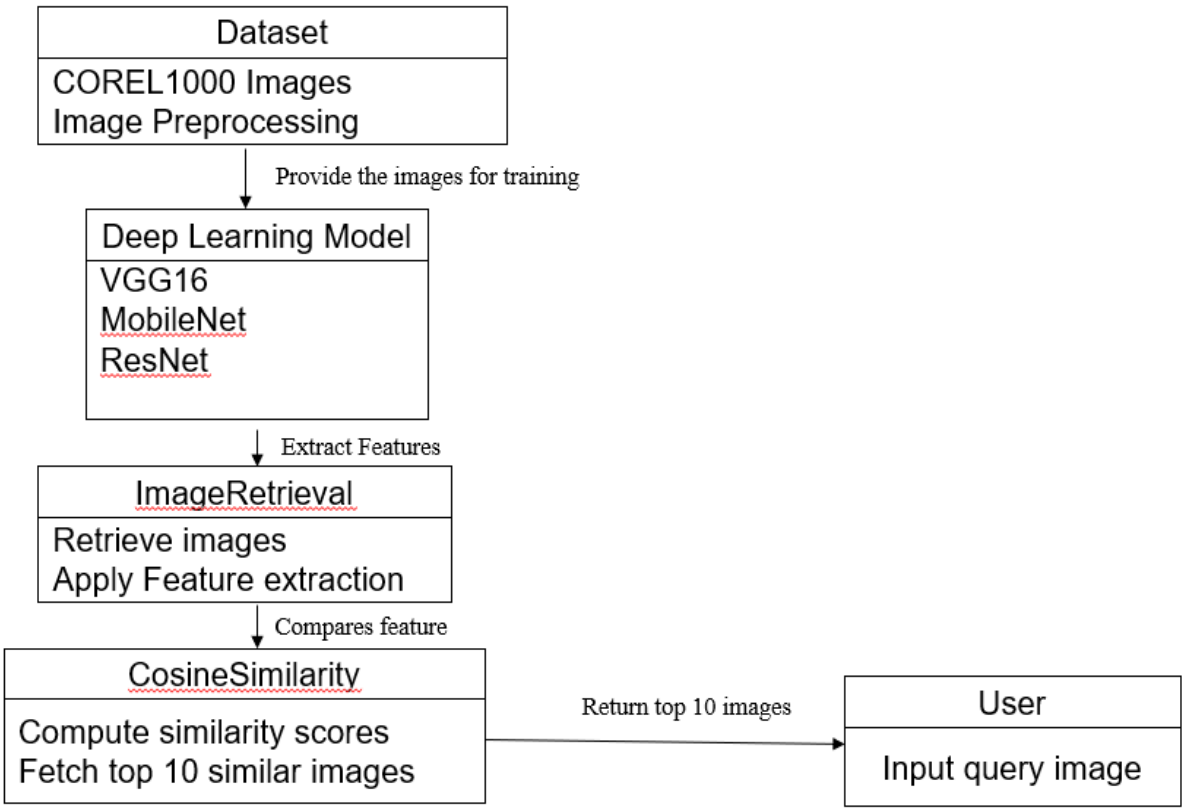


Fig 2.3.2 Class Diagram

2.3.3 Sequence Diagram

A sequence diagram outlines the interactions between objects in the detection of abnormal activities in online exams using pre-trained neural networks project to observe student behaviour for academic integrity, focusing on object lifelines and object messages. It illustrates how objects, such as the monitoring system and webcam, communicate over time through messages, enabling an understanding of the dynamic behaviour and order of operations during exams.

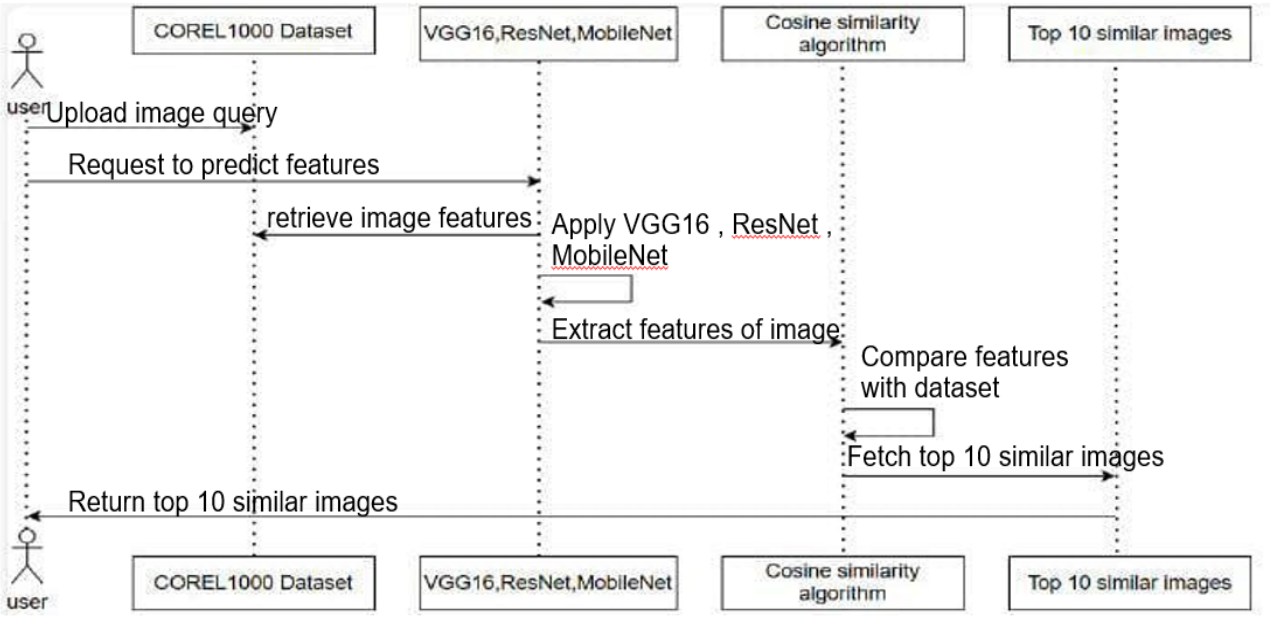
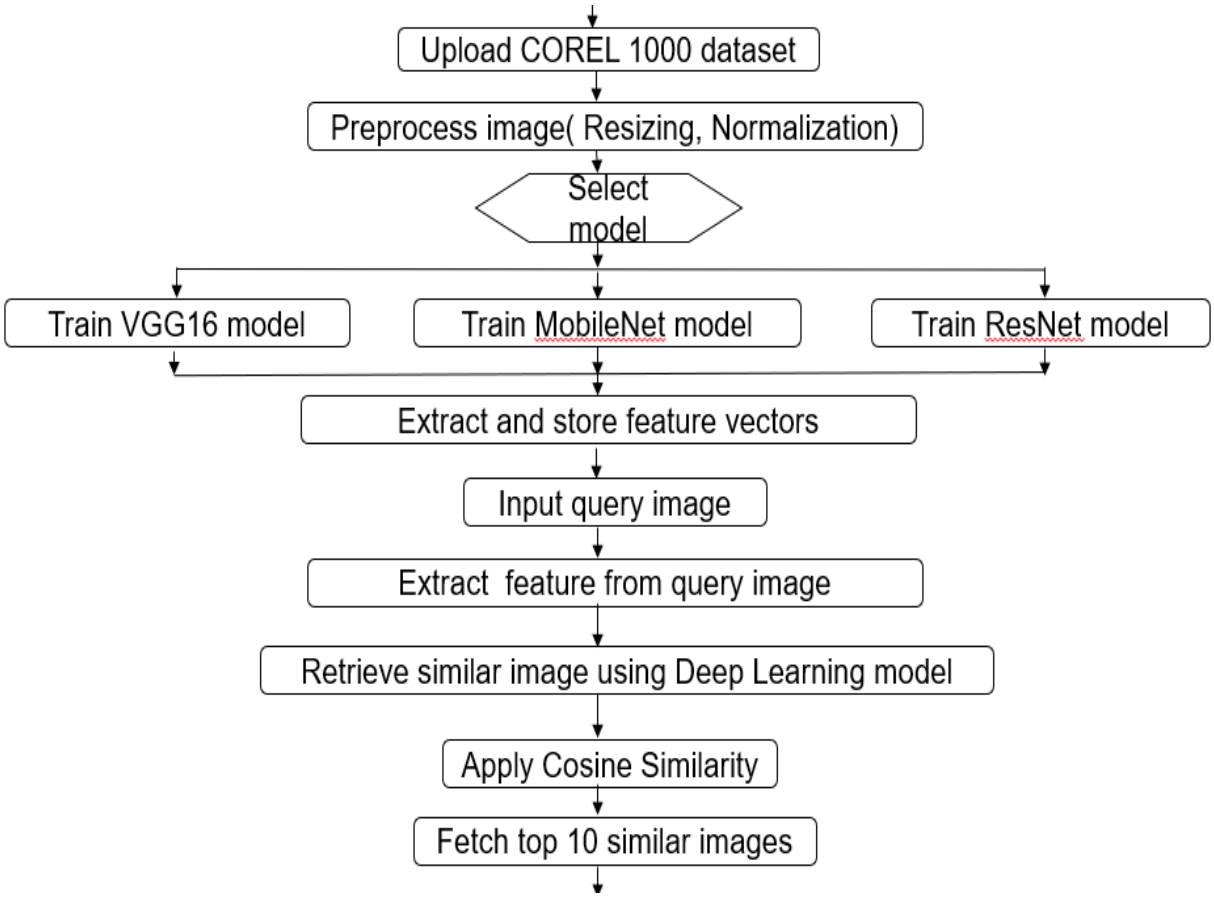


Fig 2.3.3 Sequence diagram

2.3.4 CONTROL FLOW DIAGRAM

- **Feature Extraction** involves capturing multiple types of features (color, texture, shape) and transforming the input image into a structured vector that can be compared with the database.
- **Database Search** is the core of the CBIR system, where the query image's feature vector is compared with a database of pre-extracted features.
- **Similarity Measures**: The system uses distance metrics (e.g., Euclidean distance, cosine similarity) to determine how close the query image is to the images in the database.
- **Display Results**: After ranking, the results are displayed to the user in a meaningful way, often through a UI.



2.3.4 Control flow Diagram

2.4 Functional Requirements

- Data Collection
- Data Pre-processing
- Training and Testing
- Search and Retrieval
- User Interface

2.5 Non-Functional Requirements

- Performance
- Scalability
- Accuracy
- Security
- maintainability

3.SYSTEM DESIGN

3.1 Modularization Details

- **Upload Corel Dataset** – Allows the user to upload the Corel Dataset, which is commonly used for image retrieval tasks.
- **Image Preprocessing** – Performs preprocessing steps on images, such as resizing, normalization, or feature extraction.
- **Train ResNet Algorithm** – Trains the ResNet (Residual Network) deep learning model on the dataset.
- **Train VGG16 Algorithm** – Trains the VGG16 convolutional neural network (CNN) model on the dataset.
- **Train MobileNet Algorithm** – Trains the MobileNet model, which is optimized for lightweight image processing.
- **All Algorithms Performance Graph** – Displays a performance comparison of all trained models, likely in terms of accuracy, loss, or retrieval efficiency.
- **CBIR using Test Image** – Performs Content-Based Image Retrieval using a test image, likely finding similar images from the dataset.

3.2 Algorithms

- **ResNet:**ResNet (Residual Network) is a deep convolutional neural network that helps extract powerful and rich image features. In CBIR systems, ResNet is used to generate deep feature vectors from images. These feature vectors represent the important visual patterns in images. When a user queries an image, ResNet extracts its features, compares them with the database features (using similarity measures like cosine similarity), and retrieves visually similar images. ResNet's deep layers make CBIR more accurate, even for complex and subtle image details.

- **MobileNet:**MobileNet is a lightweight deep neural network designed for fast and efficient feature extraction. In CBIR systems, MobileNet extracts compact feature vectors from images, enabling quick similarity matching. Its small size and high speed make it ideal for CBIR on mobile devices or low-power systems without sacrificing much accuracy.

- **VGG16**:VGG16 is a deep convolutional network known for its simple and uniform architecture (using 3×3 filters). In CBIR systems, VGG16 extracts rich and highly detailed feature vectors from images, which helps in retrieving images with similar content. Its depth and learned features make it effective for high-quality retrieval, though it is heavier compared to lightweight models like MobileNet.

4.IMPLEMENTATION

4.1 Coding

The project's implementation is primarily done in Python, leveraging its robust libraries for content-based image retrieval (CBIR) and computer vision tasks.

- OpenCV was used extensively for image preprocessing, feature extraction (e.g., color histograms, keypoints, and descriptors), and similarity comparisons.
- TensorFlow and Keras were employed for implementing deep learning-based feature extraction using pre-trained models like VGG16, ResNet50, and MobileNet for enhanced retrieval accuracy.
- Matplotlib and Seaborn were used for visualizing feature distributions, similarity scores, and retrieval performance metrics.
- Flask was utilized to develop a web interface where users could upload query images and receive visually similar results in real time.

```
from tkinter import messagebox
from tkinter import *
from tkinter import simpledialog
import tkinter
from tkinter import filedialog
import matplotlib.pyplot as plt
import numpy as np
from tkinter.filedialog import askopenfilename
import os
import cv2
from keras.utils.np_utils import to_categorical
from keras.layers import  MaxPooling2D
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D
from keras.models import Sequential
from keras.models import model_from_json
import pickle
from sklearn.model_selection import train_test_split
from keras.applications import ResNet50
from keras.models import Model
from keras.applications import VGG16
from keras.applications import MobileNet
from sklearn.metrics import confusion_matrix
import seaborn as sns
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
```

```
from sklearn.metrics import accuracy_score
import pandas as pd
import webbrowser
import json

main = tkinter.Tk()
main.title("Content Based Image Retrieval using Deep Learning")
main.geometry("1300x1200")

global filename
global X, Y, X_train, X_test, y_train, y_test
global accuracy, precision, recall, fscore
global vgg_classifier
labels = []

def getLabel(name):
    index = 0
    for i in range(len(labels)):
        if labels[i] == name:
            index = i
            break
    return index

def upload():
    global filename, labels
    text.delete('1.0', END)
    filename = filedialog.askdirectory(initialdir = ".")
    pathlabel.config(text=filename)
    text.insert(END,"Dataset loaded\n\n")
    for root, dirs, directory in os.walk(filename):
        for j in range(len(directory)):
            name = os.path.basename(root)
            if name not in labels:
                labels.append(name)
    text.insert(END,"Available Images class labels found in dataset: "+str(labels))

def imageProcessing():
    global filename
    global X, Y, X_train, X_test, y_train, y_test
    text.delete('1.0', END)
    if os.path.exists('model/X.txt.npy'):
        X = np.load('model/X.txt.npy')
        Y = np.load('model/Y.txt.npy')
    else:
        X = []
        Y = []
        labels.clear()
```

```
        path = filename
        for root, dirs, directory in os.walk(path):
            for j in range(len(directory)):
                name = os.path.basename(root)
                if 'Thumbs.db' not in directory[j]:
                    img = cv2.imread(root+"/"+directory[j])
                    img = cv2.resize(img, (32,32))
                    im2arr = np.array(img)
                    im2arr = im2arr.reshape(32,32,3)
                    X.append(im2arr)
                    label = getLabel(name)
                    Y.append(label)
                    print(str(j)+" "+name+" "+str(label))
        X = np.asarray(X)
        Y = np.asarray(Y)
        np.save('model/X.txt',X)
        np.save('model/Y.txt',Y)
    X = X.astype('float32')
    X = X/255
    test = X[320]
    indices = np.arange(X.shape[0])
    np.random.shuffle(indices)
    X = X[indices]
    Y = Y[indices]
    Y = to_categorical(Y)
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
    text.insert(END,"Total images found in dataset: "+str(X.shape[0])+"\n")
    text.insert(END,"Dataset train & test split details\n")
    text.insert(END,"80% images used for training & 20% images used for testing\n")
    text.insert(END,"Total images used to train Deep Learning Algorithms:
"+str(X_train.shape[0])+"\n")
    text.insert(END,"Total images used to test Deep Learning Algorithms: "+str(X_test.shape[0])+"\n")
    cv2.imshow("Processed Sample Image",cv2.resize(test,(200,200)))
    cv2.waitKey(0)

#testprediction function to calculate metrics
def testPrediction(name, classifier, X_test, y_test):
    global labels
    predict = classifier.predict(X_test) #perform prediction using VGG, Resnet or mobilenet classifier
    predict = np.argmax(predict, axis=1)
    testY = np.argmax(y_test, axis=1)
    p = precision_score(testY, predict,average='macro') * 100 #calculate precision and other metrics
    r = recall_score(testY, predict,average='macro') * 100
    f = f1_score(testY, predict,average='macro') * 100
    a = accuracy_score(testY,predict)*100
    text.insert(END,"Test Images Size: "+str(X_test.shape[0])+"\n")
    text.insert(END,name+' Accuracy  : '+str(a)+"\n")
```

```
    text.insert(END,name+' Precision : '+str(p)+"\n")
    text.insert(END,name+' Recall    : '+str(r)+"\n")
    text.insert(END,name+' FMeasure  : '+str(f)+"\n")
    conf_matrix = confusion_matrix(testY, predict)
    FP = conf_matrix.sum(axis=0) - np.diag(conf_matrix)
    TP = np.diag(conf_matrix)
    FN = 180 - (TP + FP)
    TN = 180 - (TP + FP)
    text.insert(END,name+" TP = "+str(np.sum(TP) / X_test.shape[0])+" FP = "+str(np.sum(FP) /
X_test.shape[0])+" TN = 0 FN = 0\n\n")
    accuracy.append(a)
    precision.append(p)
    recall.append(r)
    fscore.append(f)
    sns.heatmap(conf_matrix, xticklabels = labels, yticklabels = labels, annot = True, cmap="viridis"
,fmt ="g");
    plt.title(name+" Confusion matrix")
    plt.show()


def runResnet():
    global X_train, X_test, y_train, y_test
    text.delete('1.0', END)
    global accuracy, precision, recall, fscore, jaccard, error_rate, auc
    accuracy = []
    precision = []
    recall = []
    fscore = []
    jaccard = []
    error_rate = []
    auc = []
    if os.path.exists('model/resnet_model.json'):
        with open('model/resnet_model.json', "r") as json_file:
            loaded_model_json = json_file.read()
            classifier = model_from_json(loaded_model_json)
        json_file.close()
        classifier.load_weights("model/resnet_model_weights.h5")
        classifier._make_predict_function()
    else:
        #defining RESNET MODEL
        resnet = ResNet50(include_top=False, weights='imagenet', input_shape=(32, 32, 3))
        for layer in resnet.layers:
            layer.trainable = False
        classifier = Sequential()
        classifier.add(resnet)
        classifier.add(Convolution2D(32, 1, 1, input_shape = (X_train.shape[1], X_train.shape[2],
X_train.shape[3]), activation = 'relu'))
```

```
        classifier.add(MaxPooling2D(pool_size = (1, 1)))
        classifier.add(Convolution2D(32, 1, 1, activation = 'relu'))
        classifier.add(MaxPooling2D(pool_size = (1, 1)))
        classifier.add(Flatten())
        classifier.add(Dense(output_dim = 256, activation = 'relu'))
        classifier.add(Dense(output_dim = y_train.shape[1], activation = 'softmax'))
        classifier.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
        hist = classifier.fit(X, Y, batch_size=16, epochs=20, shuffle=True, verbose=2,
validation_data=(X_test, y_test))
        classifier.save_weights('model/resnet_model_weights.h5')
        model_json = classifier.to_json()
        with open("model/resnet_model.json", "w") as json_file:
            json_file.write(model_json)
        f = open('model/resnet_history.pckl', 'wb')
        pickle.dump(hist.history, f)
        f.close()
    print(classifier.summary())
    print(str(X_test.shape)+" "+str(y_test.shape))
    testPrediction("Resnet Algorithm", classifier, X_test, y_test)

def runVGG():
    global vgg_classifier
    if os.path.exists('model/vgg_model.json'):
        with open('model/vgg_model.json', "r") as json_file:
            loaded_model_json = json_file.read()
            classifier = model_from_json(loaded_model_json)
        json_file.close()
        classifier.load_weights("model/vgg_model_weights.h5")
        classifier._make_predict_function()
        classifier.layers.pop()
        layer = classifier.layers[-2]
        vgg_classifier = Model(inputs=classifier.input, outputs=layer.output)
    else:
        #defining VGG model
        vgg = VGG16(input_shape=(X_train.shape[1], X_train.shape[2], X_train.shape[3]),
include_top=False, weights="imagenet")
        vgg.trainable = False
        classifier = Sequential()
        classifier.add(vgg)
        classifier.add(Convolution2D(32, 1, 1, input_shape = (X_train.shape[1], X_train.shape[2],
X_train.shape[3]), activation = 'relu'))
        classifier.add(MaxPooling2D(pool_size = (1,1)))
        classifier.add(Convolution2D(32, 1, 1, activation = 'relu'))
        classifier.add(MaxPooling2D(pool_size = (1, 1)))
        classifier.add(Flatten())
        classifier.add(Dense(output_dim = 256, activation = 'relu'))
        classifier.add(Dense(output_dim = y_train.shape[1], activation = 'softmax'))
```

```
        classifier.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
        hist = classifier.fit(X, Y, batch_size=16, epochs=20, shuffle=True, verbose=2,
validation_data=(X_test, y_test))
        classifier.save_weights('model/vgg_model_weights.h5')
        model_json = classifier.to_json()
        with open("model/vgg_model.json", "w") as json_file:
            json_file.write(model_json)
        json_file.close()
        f = open('model/vgg_history.pckl', 'wb')
        pickle.dump(hist.history, f)
        f.close()
    print(classifier.summary())
    print(str(X_test.shape)+" "+str(y_test.shape))
    testPrediction("VGG16 Algorithm", classifier, X_test, y_test)

def runMobilenet():
    if os.path.exists('model/mobilenet_model.json'):
        with open('model/mobilenet_model.json', "r") as json_file:
            loaded_model_json = json_file.read()
            classifier = model_from_json(loaded_model_json)
        json_file.close()
        classifier.load_weights("model/mobilenet_model_weights.h5")
        classifier._make_predict_function()
    else:
        #defining mobilenet object
        mobilenet = MobileNet(input_shape = (32, 32, 3), include_top = False, weights = "imagenet")
        #last google net model layer will be ignore to concatenate banana custom model
        inception.trainable = False
        classifier = Sequential()
        #adding mobilenet features to our model
        classifier.add(mobilenet)
        #defining CNN layer with 32 filters to filter image features
        classifier.add(Convolution2D(32, 1, 1, input_shape = (X_train.shape[1], X_train.shape[2],
X_train.shape[3]), activation = 'relu'))
        #using max pooling we will collect all filter features
        classifier.add(MaxPooling2D(pool_size = (1,1)))
        classifier.add(Convolution2D(32, 1, 1, activation = 'relu'))
        classifier.add(MaxPooling2D(pool_size = (1, 1)))
        classifier.add(Flatten())
        classifier.add(Dense(output_dim = 256, activation = 'relu'))
        classifier.add(Dense(output_dim = y_train.shape[1], activation = 'softmax'))
        #compiling the model
        classifier.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
        #start training model
        hist = classifier.fit(X, Y, batch_size=16, epochs=20, shuffle=True, verbose=2,
validation_data=(X_test, y_test))
```

```
        classifier.save_weights('model/mobilenet_model_weights.h5')#save the weights for future
used
        model_json = classifier.to_json()
        with open("model/mobilenet_model.json", "w") as json_file:
            json_file.write(model_json)
        json_file.close()
        f = open('model/mobilenet_history.pckl', 'wb')
        pickle.dump(hist.history, f)
        f.close()
    print(classifier.summary())
    #calling testPrediction function to perform prediction using mobilenet on test data and then
calculate accuracy and other metrics
    testPrediction("MobileNet Algorithm", classifier, X_test, y_test)

def graph():
    output = "<table align=center border=1><tr><th>Algorithm
Name</th><th>Precision</th><th>Recall</th><th>F1 SCORE</th><th>Accuracy</th></tr>"
    columns = ["Algorithm Name","Precison","Recall","FScore","Accuracy"]
    values = []
    algorithm_names = ["Resnet","VGG16","MobileNet"]
    for i in range(len(algorithm_names)):
        output += "<tr><td>"+algorithm_names[i]+"</td>"
        output += "<td>"+str(precision[i])+"</td>"
        output += "<td>"+str(recall[i])+"</td>"
        output += "<td>"+str(fscore[i])+"</td>"
        output += "<td>"+str(accuracy[i])+"</td></tr>"
    f = open("output.html", "w")
    f.write(output)
    f.close()
    webbrowser.open("output.html",new=1)
    df = pd.DataFrame([['Resnet','Precision',precision[0]],['Resnet','Recall',recall[0]],['Resnet','F1
Score',fscore[0]],['Resnet','Accuracy',accuracy[0]],
                       ['VGG16','Precision',precision[1]],['VGG16','Recall',recall[1]],['VGG16','F1
Score',fscore[1]],['VGG16','Accuracy',accuracy[1]],
                       ['MobileNet','Precision',precision[2]],['MobileNet','Recall',recall[2]],['MobileNet','F1
Score',fscore[2]],['MobileNet','Accuracy',accuracy[2]],

                      ],columns=['Parameters','Algorithms','Value'])
    df.pivot("Parameters", "Algorithms", "Value").plot(kind='bar')
    plt.title("Resnet, VGG16, & MobileNet Performance Graph")
    plt.show()

def close():
    main.destroy()

def format_image_for_display(img_path):
    img = cv2.imread(img_path)
```

```
    img = cv2.resize(img, (300, 300))
    return img

def read_and_format_image(img_path):
    img = cv2.imread(img_path)
    resized_img = cv2.resize(img, (32, 32))
    return resized_img.reshape((1, 32, 32, 3))

def cosineBasedimageRetrieval(query, vgg_classifier): #cosine based image similarity calculation
    dataset = json.load(open('feat_vectors.json'))
    distances = []
    for key in list(dataset.keys()):
        dico = {}
        img = dataset[key]
        cosine_similarity = np.linalg.norm(np.array(query) - np.array(img))
        print(cosine_similarity)
        if cosine_similarity < 5:
            dico['img_name'] = key
            dico['distance'] = cosine_similarity
            distances.append(dico)
    distances.sort(key=lambda x: x['distance'])
    images = [d['img_name'] for d in distances]
    list_images = []
    for i in range(len(images)):
        print(images[i])
        list_images.append(format_image_for_display(images[i]))
    return list_images

def cbir():
    text.delete('1.0', END)
    global vgg_classifier
    filename = askopenfilename(initialdir = "queryImages") #uploading query image
    pathlabel.config(text=filename)
    text.insert(END,filename+" loaded\n")
    input_img = read_and_format_image(filename) #reading image from given path
    query = list(vgg_classifier.predict(input_img / 255).flatten().astype(float))#using VGG16 we are
predicting or extracting features for given input image
    list_images  = cosineBasedimageRetrieval(query, vgg_classifier)#now calling cosine base function
to calculate similariy between images to perform retrieval
    temp = []
    for i in range(len(list_images)):
        temp.append(list_images[i])
        if i > 10:
            break
    list1 = []
    list2 = []
    list3 = []
```

```
    list1.append(temp[0])
    list1.append(temp[1])
    list1.append(temp[2])
    list2.append(temp[3])
    list2.append(temp[4])
    list2.append(temp[5])
    list3.append(temp[6])
    list3.append(temp[7])
    list3.append(temp[8])
    list3.append(temp[9])
    stack1 = cv2.hconcat(list1)
    stack2 = cv2.hconcat(list2)
    stack3 = cv2.hconcat(list3)
    query = format_image_for_display(filename)
    cv2.imshow("Original Image",query)
    cv2.imshow('Retrieved images 1 to 3', stack1)
    cv2.imshow('Retrieved images 4 to 6', stack2)
    cv2.imshow('Retrieved images 6 to 10', stack3)
    cv2.waitKey(0)

def close():
    main.destroy()

font = ('times', 16, 'bold')
title = Label(main, text='Content Based Image Retrieval using Deep Learning')
title.config(bg='brown', fg='white')
title.config(font=font)
title.config(height=3, width=120)
title.place(x=0,y=5)

font1 = ('times', 13, 'bold')
upload = Button(main, text="Upload Coral Dataset", command=upload)
upload.place(x=50,y=100)
upload.config(font=font1)

pathlabel = Label(main)
pathlabel.config(bg='brown', fg='white')
pathlabel.config(font=font1)
pathlabel.place(x=350,y=100)

preprocess = Button(main, text="Image Preprocessing", command=imageProcessing)
preprocess.place(x=50,y=150)
preprocess.config(font=font1)

resnetmodel = Button(main, text="Train Resnet Algorithm", command=runResnet)
resnetmodel.place(x=300,y=150)
resnetmodel.config(font=font1)
```

```
vggmodel = Button(main, text="Train VGG16 Algorithm", command=runVGG)
vggmodel.place(x=550,y=150)
vggmodel.config(font=font1)

mnmodel = Button(main, text="Train MobileNet Algorithm", command=runMobilenet)
mnmodel.place(x=810,y=150)
mnmodel.config(font=font1)

graphButton = Button(main, text="All Algorithms Performance Graph", command=graph)
graphButton.place(x=50,y=200)
graphButton.config(font=font1)

cbirButton = Button(main, text="CBIR using Test Image", command=cbir)
cbirButton.place(x=380,y=200)
cbirButton.config(font=font1)

exitButton = Button(main, text="Exit", command=close)
exitButton.place(x=810,y=200)
exitButton.config(font=font1)

font1 = ('times', 12, 'bold')
text=Text(main,height=30,width=150)
scroll=Scrollbar(text)
text.configure(yscrollcommand=scroll.set)
text.place(x=10,y=250)
text.config(font=font1)

main.config(bg='brown')
main.mainloop()
```

4.2 ScreenShots

The following images display the User Interface (UI) of the “content based image retrieval” system, launched in a web-browser.

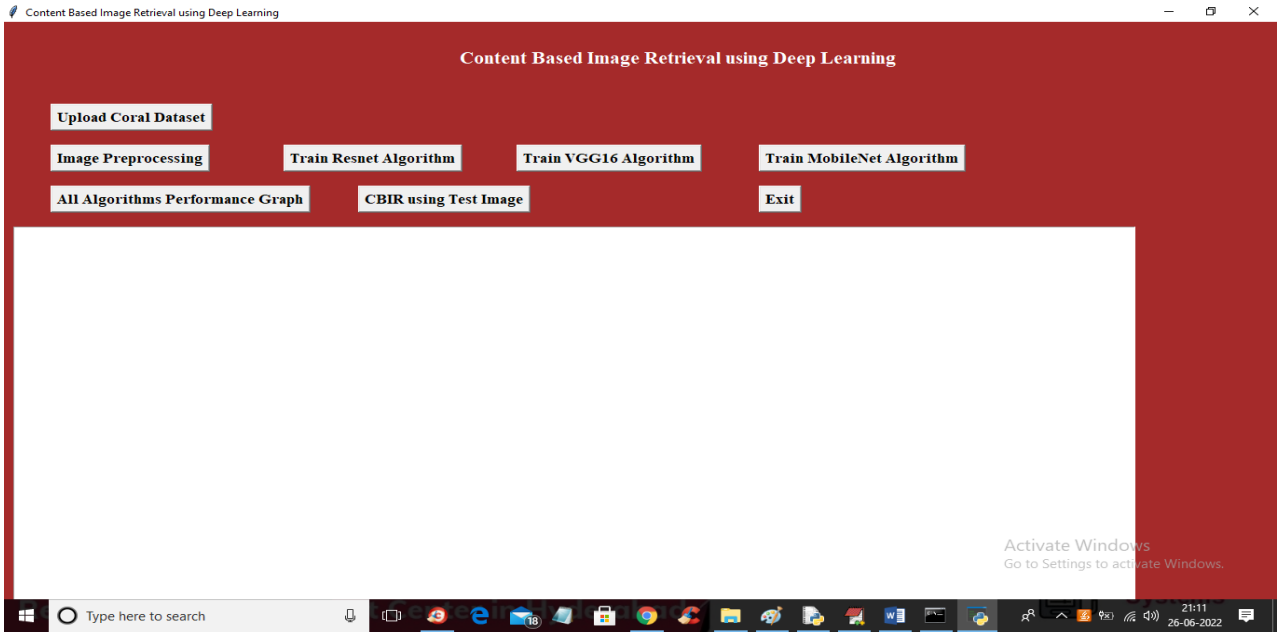


Fig 4.2.1 Home Page

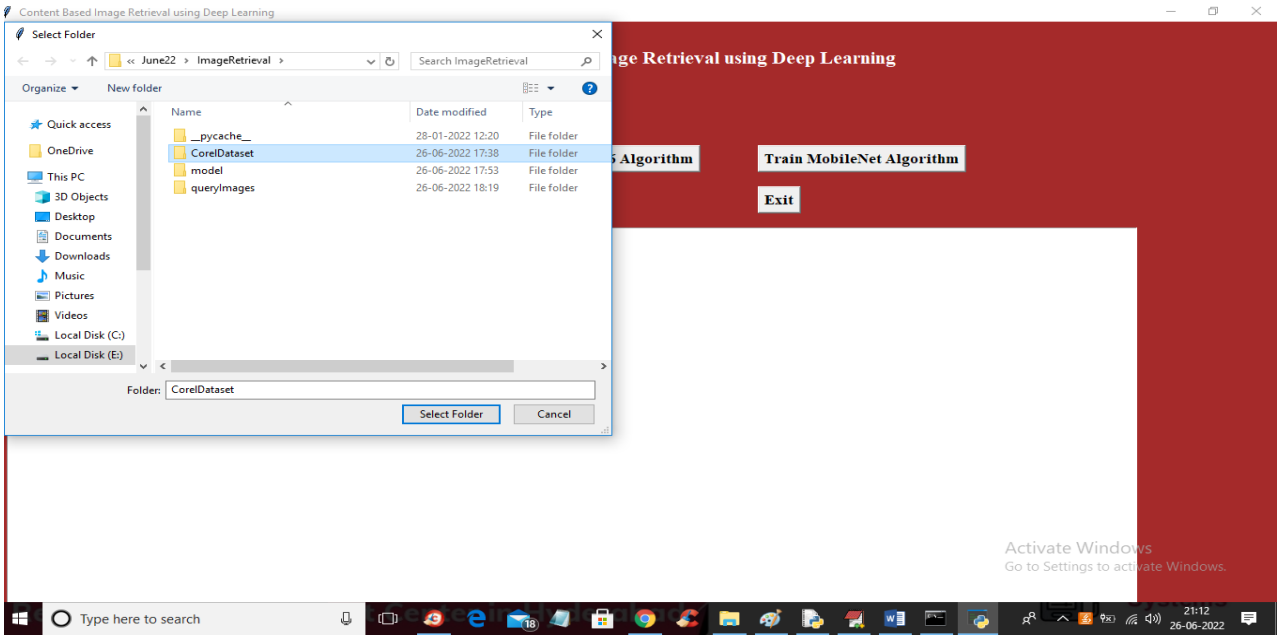


Fig.4.2.2 Upload Corel Dataset

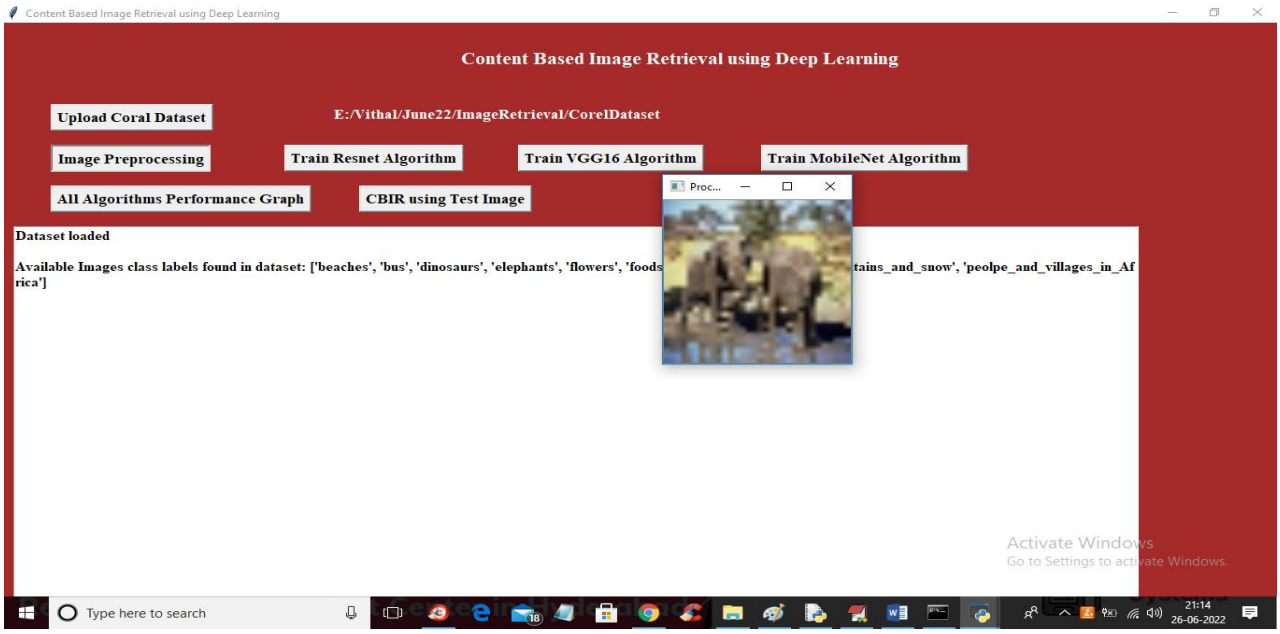


Fig. 4.2.3 Image Preprocessing

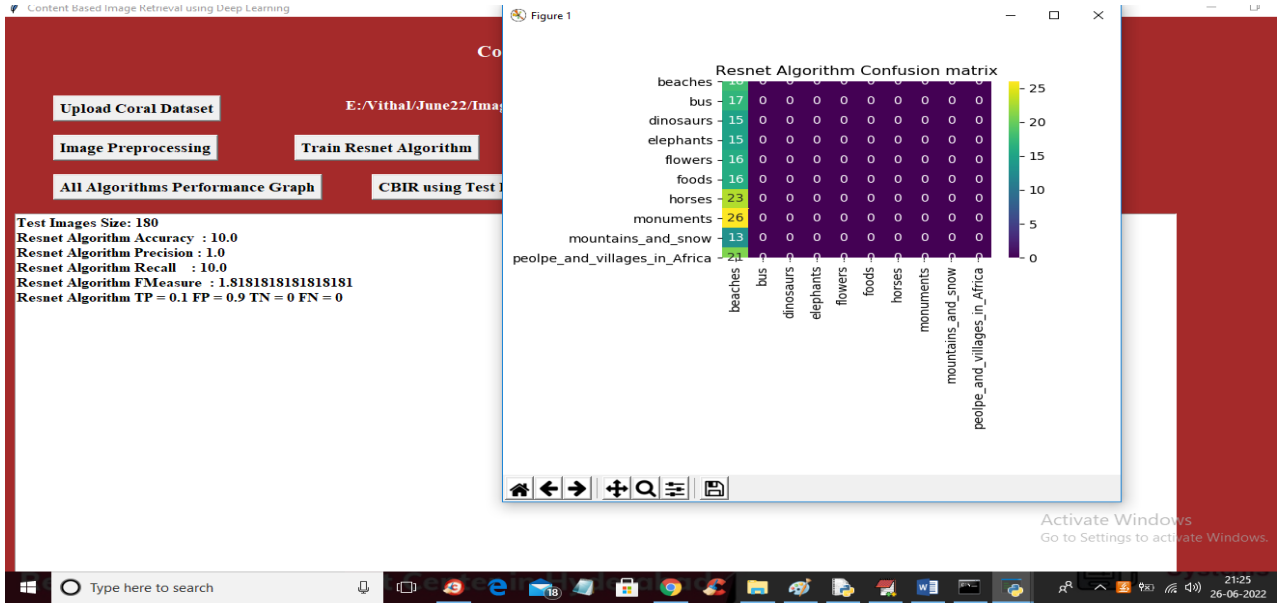


Fig 4.2.4 Train ResNet Algorithm

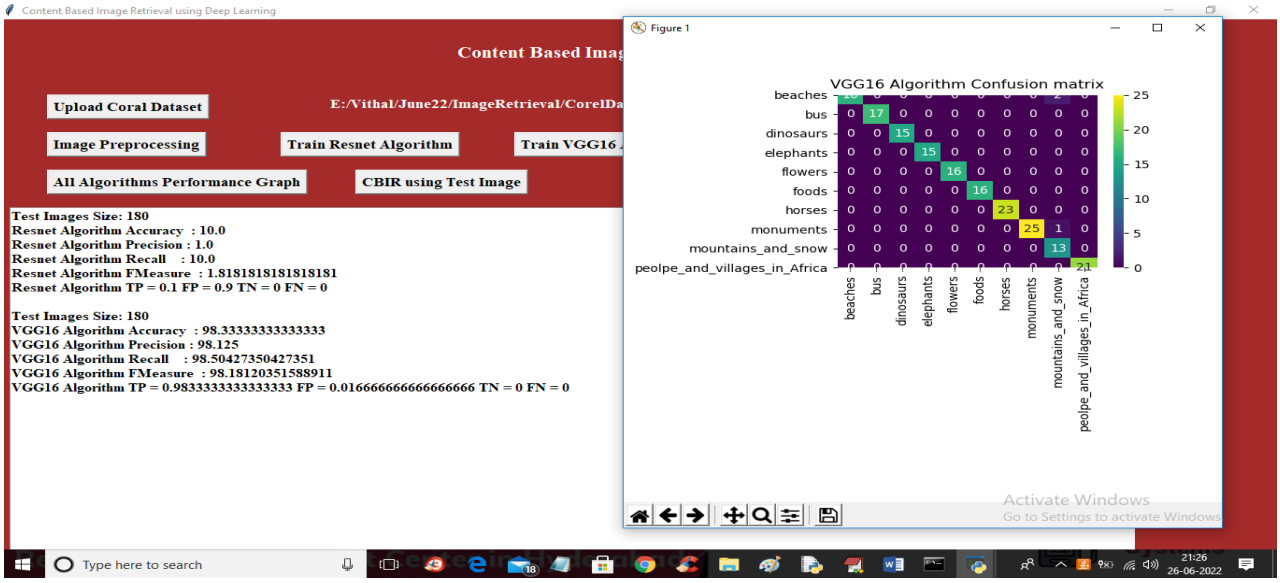


Fig 4.2.5 Train VGG16 Algorithm

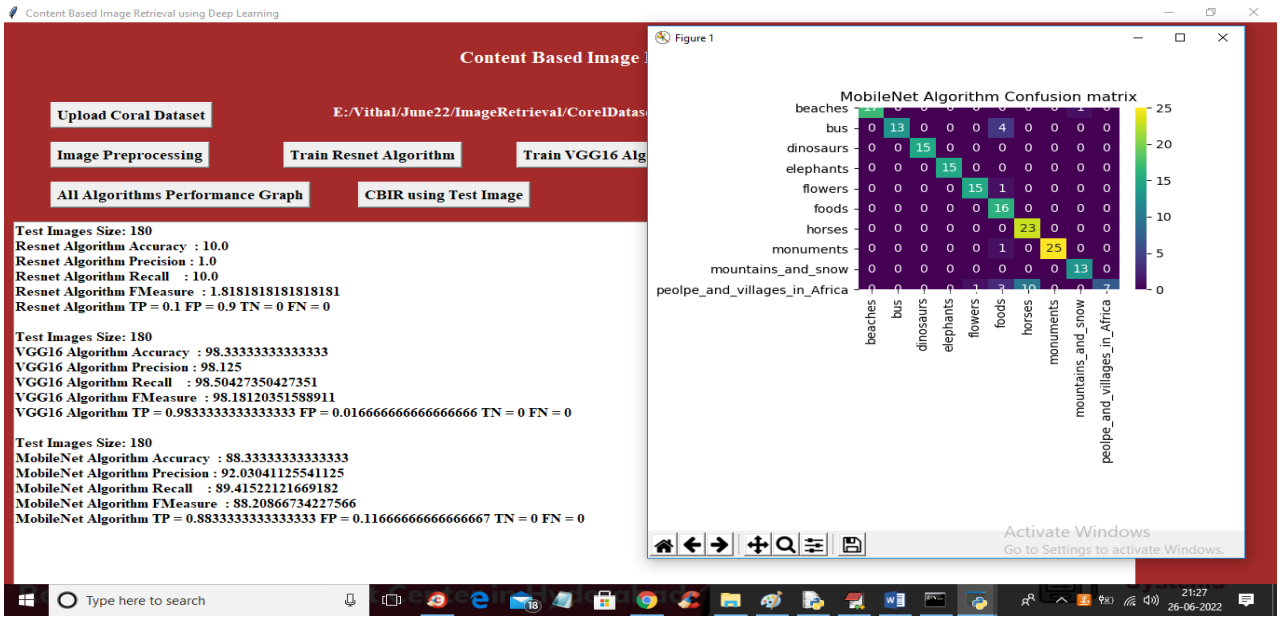


Fig 4.2.6 Train MobileNet Algorithm

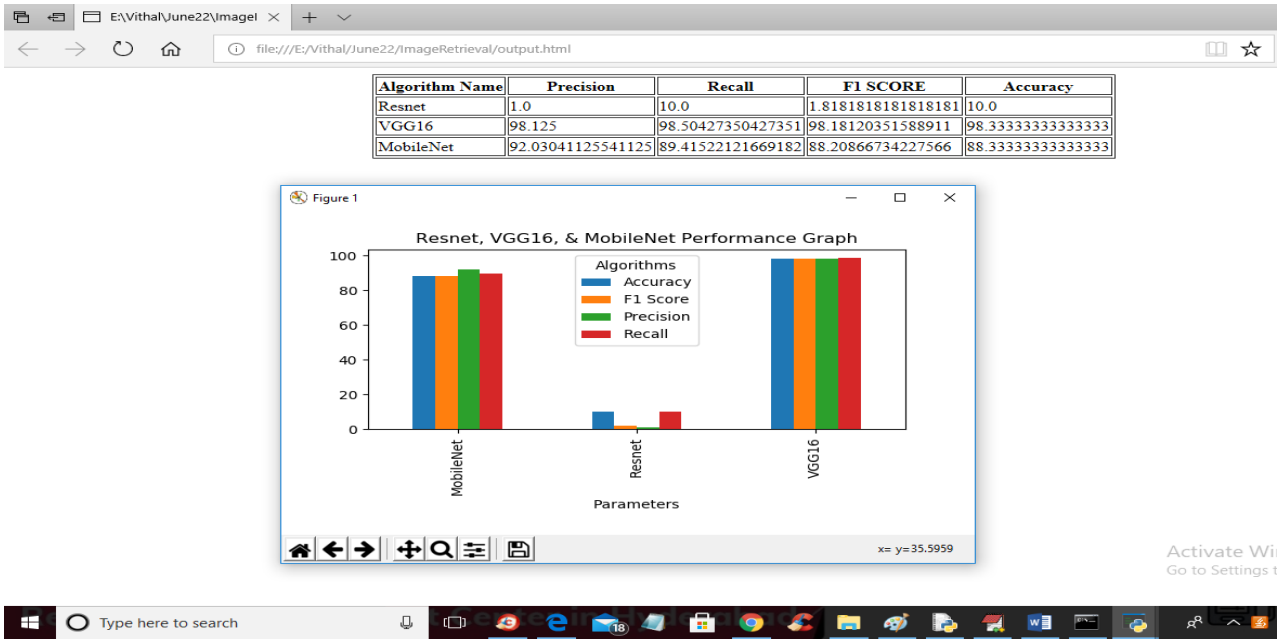


Fig 4.2.7Algorithm Performance Graph

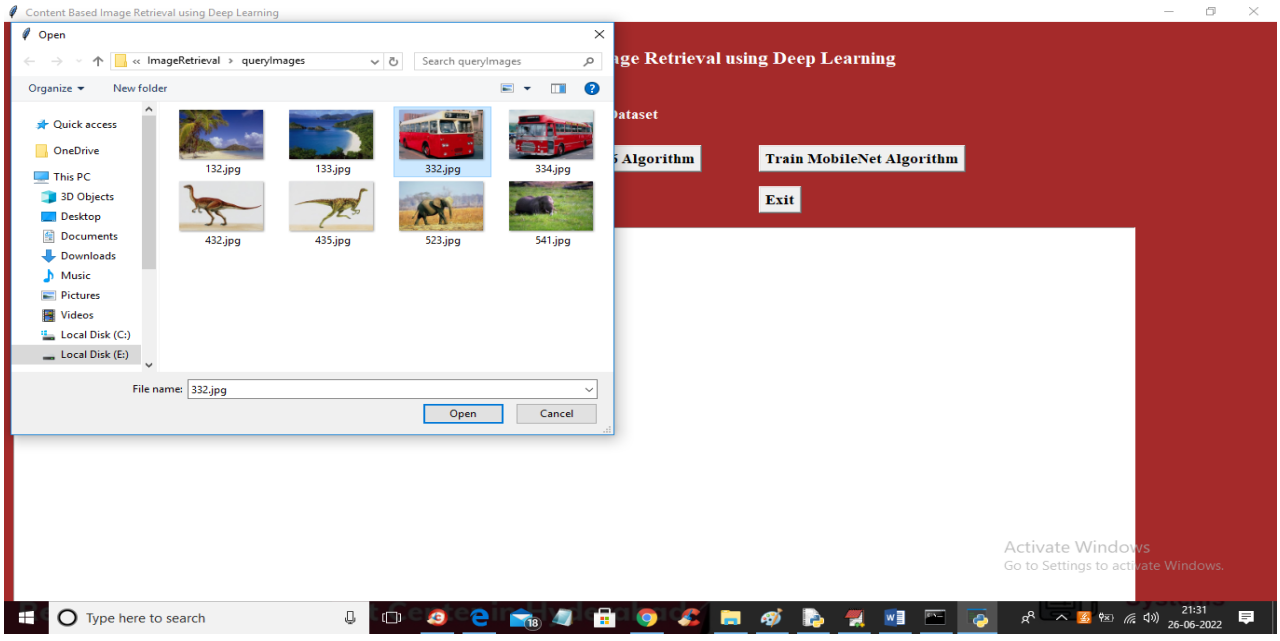


Fig 4.2.8 Upload Query Image

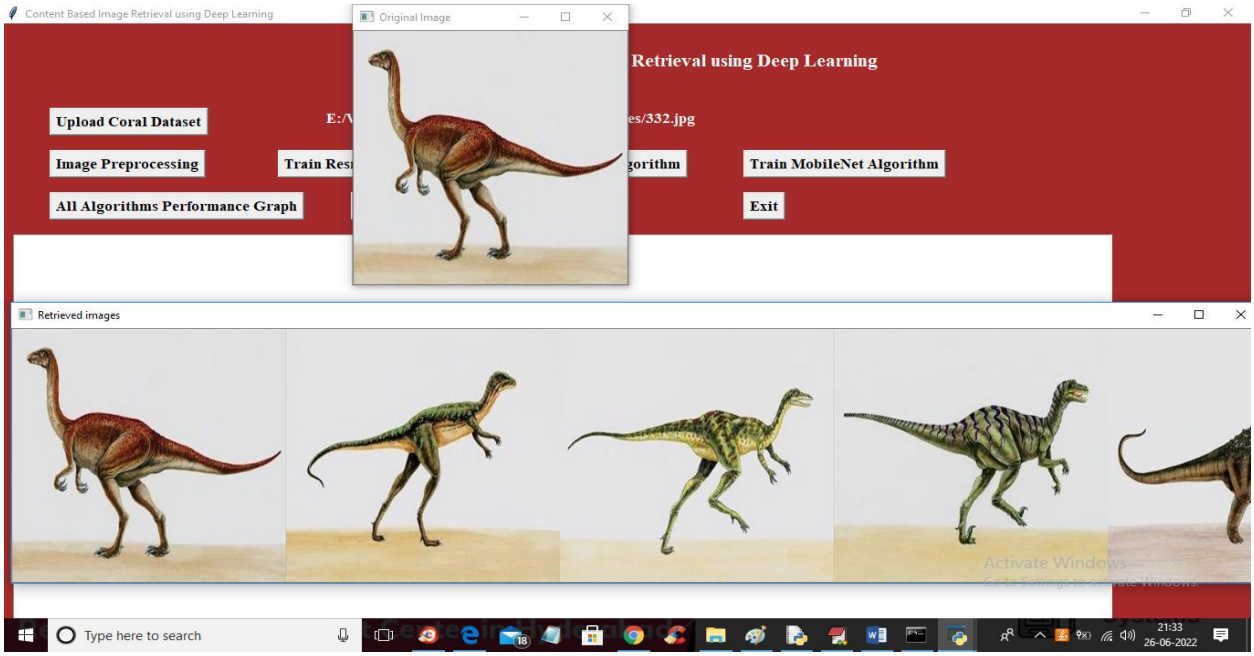


Fig. 4.2.9 Retrieving similar images

4.3 Test Cases

Test cases for a CBIR system check if the system accurately retrieves visually similar images. Common tests include querying known and unknown images, verifying feature extraction, and evaluating similarity scores and response time.

Sno	Test case Description	Input given	Expected output	Actual output	Result
1	Data Collection	Collection of images separated into classes	Corel dataset is created	Corel dataset is created	pass
2	UI display	Select dataset file	Available Images class labels found in dataset	Available Images class labels found in dataset	Pass
3	Data Preprocessing	Corel dataset	Dataset is cleaned and preprocessed	Dataset is cleaned and preprocessed	Pass
4	Deep learning model training	Preprocessed Corel dataset	80% data used for training and 20% data used for testing	80% data used for training and 20% data used for testing	pass

5	Model Evaluation	Preprocessed Corel dataset	Accuracy : 8.33333333333333 Precision : 0.83333333333333 Recall : 10.0	Accuracy : 8.3333333333333 Precision : 0.8333333333333 Recall : 10.0	Pass
6	Performance comparison	Accuracy : 8.3333333333333 Precision : 0.8333333333333 Recall : 10.0	Performance Comparison Graph generated	Performance Comparison Graph generated	Pass
7	Image retrieval	User selected query image	Similar images retrieved	Similar images retrieved	Pass

4.3.1 Test Cases Analysis

4.4 CONCLUSION

This project presents a Content-Based Image Retrieval (CBIR) system that enhances the accuracy and efficiency of retrieving visually similar images by integrating deep learning and similarity measurement techniques. By leveraging state-of-the-art deep learning models such as VGG16, MobileNet, and ResNet for feature extraction, the system effectively captures intricate image details and patterns. The application of COSINE similarity further refines the retrieval process, ensuring precise similarity measurement and ranking of the top 10 most relevant images.

The proposed approach demonstrates significant improvements over traditional text-based and conventional CBIR methods by offering a more intelligent, scalable, and automated solution for image retrieval. This makes it highly applicable in various domains, including multimedia search, digital libraries, medical imaging, and recommendation systems.

By combining deep learning with similarity metrics, this system sets a foundation for future advancements in efficient and intelligent image retrieval, paving the way for more sophisticated applications in computer vision and artificial intelligence..

.

4.5 FUTURE ENHANCEMENT

In the future, the CBIR system can be improved by integrating deep learning models like Convolutional Neural Networks (CNNs) for more accurate and efficient image retrieval. Adding a relevance feedback mechanism would allow the system to learn from user interactions and continuously refine results. To handle large datasets, scalable indexing techniques such as FAISS or cloud-based solutions can be implemented. Furthermore, combining multiple types of features — such as color, texture, and semantic information — can make the system more robust. Expanding the system to support real-time retrieval, multi-modal search (image + text), and mobile or web deployment would significantly enhance user experience and broaden application areas.

REFRENCES

1. User Interfaces in C#: Windows Forms and Custom Controls by Matthew MacDonald.
2. Applied Microsoft® .NET Framework Programming (Pro-Developer) by Jeffrey Richter.
3. Practical .Net2 and C#2: Harness the Platform, the Language, and the ramework by Patrick Smacchia.
4. Data Communications and Networking, by Behrouz A Forouzan.
5. Computer Networking: A Top-Down Approach, by James F. Kurose.
6. Operating System Concepts, by Abraham Silberschatz.
7. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “Above the clouds: A berkeley view of cloud computing,” University of California, Berkeley, Tech. Rep. USB-EECS-2009-28, Feb 2009.