A REPORT OF TRAINING

at

Defence Research and Development Organisation

Defence Scientific Information and Documentation Centre (Desidoc)

Submitted by
AKSHIT SEHGAL
(Maharaja Surajmal Institute of Technology)

HARNOOR SINGH CHAWLA
(Vivekananda Institute of Professional Studies)

Image based search engine with CNN and transfer learning

Aim - Our aim is to build an image based search engine. We want the user to give an image and based on the image the model will present similar images from the database.

We started our research from pyimagesearch

(https://www.pyimagesearch.com/2014/12/01/complete-guide-building-image-search-engine-python-opencv/), where we got an idea about how we're going to move ahead with the model.

There we saw that we can make our model using openCV, further we saw that there are three types of image search engine:

1. Search by Meta-Data,

2. Search by Example,

3. Hybrid Approach of the two.

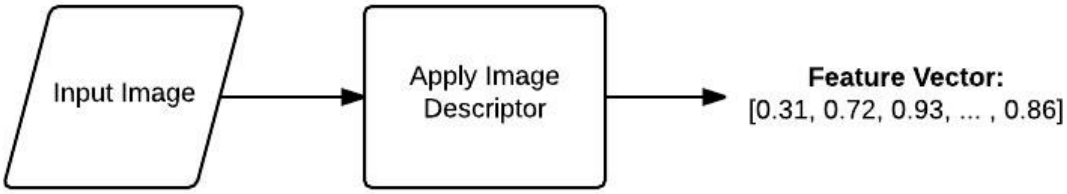
We further learnt about CBIR(Content based Image Retrieval).

Then we searched further on other websites like Github, where we found that we can also use transfer learning for making our model, which we did use in our model.

Below given are the details of our model:

What is an Image based search engine?

Image based search engines that quantify the contents of an image are called Content-Based Image Retrieval (CBIR) systems. The term CBIR is commonly used in the academic literature, but in reality, it's simply a fancier way of saying "image search engine", with the added poignancy that the search engine is relying strictly on the contents of the image and not any textual annotations associated with the image.



In the above image we can see that the user gives an image that he/she wants to search and find similar images of.

This is broadly what we want our app to do:

1. Takes an Input image from the user.
2. System starts the preprocessing of the image with the help of our app.
3. The image is then passed through our model which we build using transfer learning. We will experiment with various pre built models for image classification from tf.keras.applications
4. The feature extractor model will process the image and return a feature vector for the image.
5. The user's image feature vector will be compared with all the image feature vectors in the dataset.
6. We will use cosine similarity to compare the similarity in feature vectors to get similar images. refer image_cosine_similarity, cosine_similarity
7. The model will return all the similar images and will display it in the web app.

DATA

Dataset: We''ll use the flicker_8k(https://www.kaggle.com/ming666/flicker8k-dataset) dataset from kaggle.com. The dataset contains 8099 random unlabeled images which we'll be using in our model to search similar images from.

Data preprocessing: Our data is raw images and we have to preprocess it before passing into the model. We will define a function that'll mainly do the following:

1. The function will take the directory name, image_shape, scale as the arguments.
2. It'll read the data from the directory
3. It'll convert the data into tensors or arrays
4. Then it will resize the image into the shape given in the argument(default is (224,244,3))
5. If scale is True then the image will be normalized else will remain the same(Some models like EfficientNet have rescaling in-built and don't require normalization)
6. It'll return the preprocessed image.

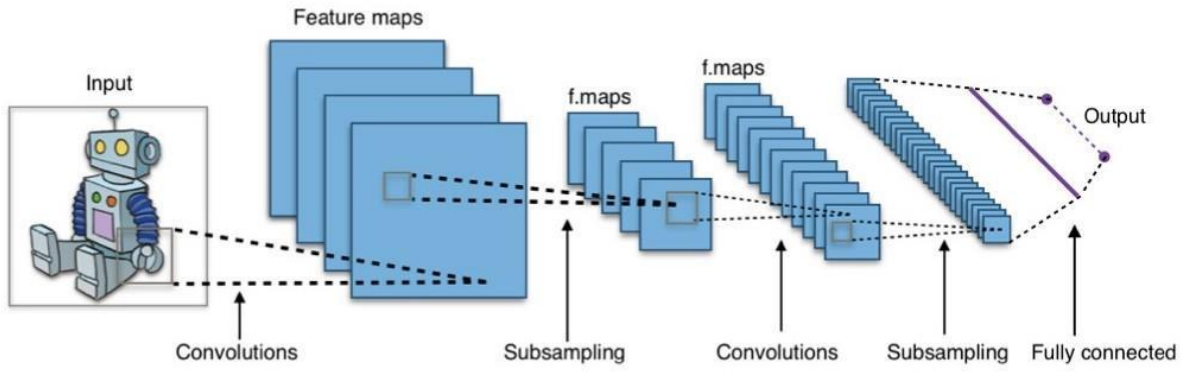
🔍Transfer Learning Model

There are two main benefits to using transfer learning:

1. Can leverage an existing neural network architecture proven to work on problems similar to our own.
2. Can leverage a working neural network architecture which has **already learned** patterns on similar data to our own. This often results in achieving great results with less custom data.

What this means is, instead of hand-crafting our own neural network architectures or building them from scratch, we can utilize models which have worked for others.

And instead of training our own models from scratch on our own datasets, we can take the patterns a model has learned from datasets such as ImageNet (millions of images of different objects) and use them as the foundation of our own. Doing this often leads to getting great results with less
data.



Convolution Neural Network

Source: Wikipedia

🧠Layers in model:

Model contains of 3 layers:

1.**INPUT LAYER**

This layer takes the images in our dataset as input having the shape (224,224,3)

2.**FUNCTIONAL LAYER**

This is the main layer of the model which contains the state of the art CNN model which we take from tf.keras.applications. For example:-
EfficientNetB0,ResNet50,VGG16,etc. This layer will learn the features in the images necessary for classification. We'll build a feature-extractor transfer learning model from this which will learn patterns in our image data.

3.**GLOBAL_AVERAGE_POOL LAYER**

This layer will pool the output from the FUNCTIONAl LAYER and give us our pooled feature vector which we will use to compare different images from the database.

🔍Cosine Similarity

To check if two feature vectors are similar we will use cosine similarity function from sklearn.metrics.pairwise.cosine_similarity.

Cosine similarity compares two feature vectors and outputs how similar two feature vectors are therefore telling how similar two images are

Cosine similarity, or the cosine kernel, computes similarity as the normalized dot product of X and Y:

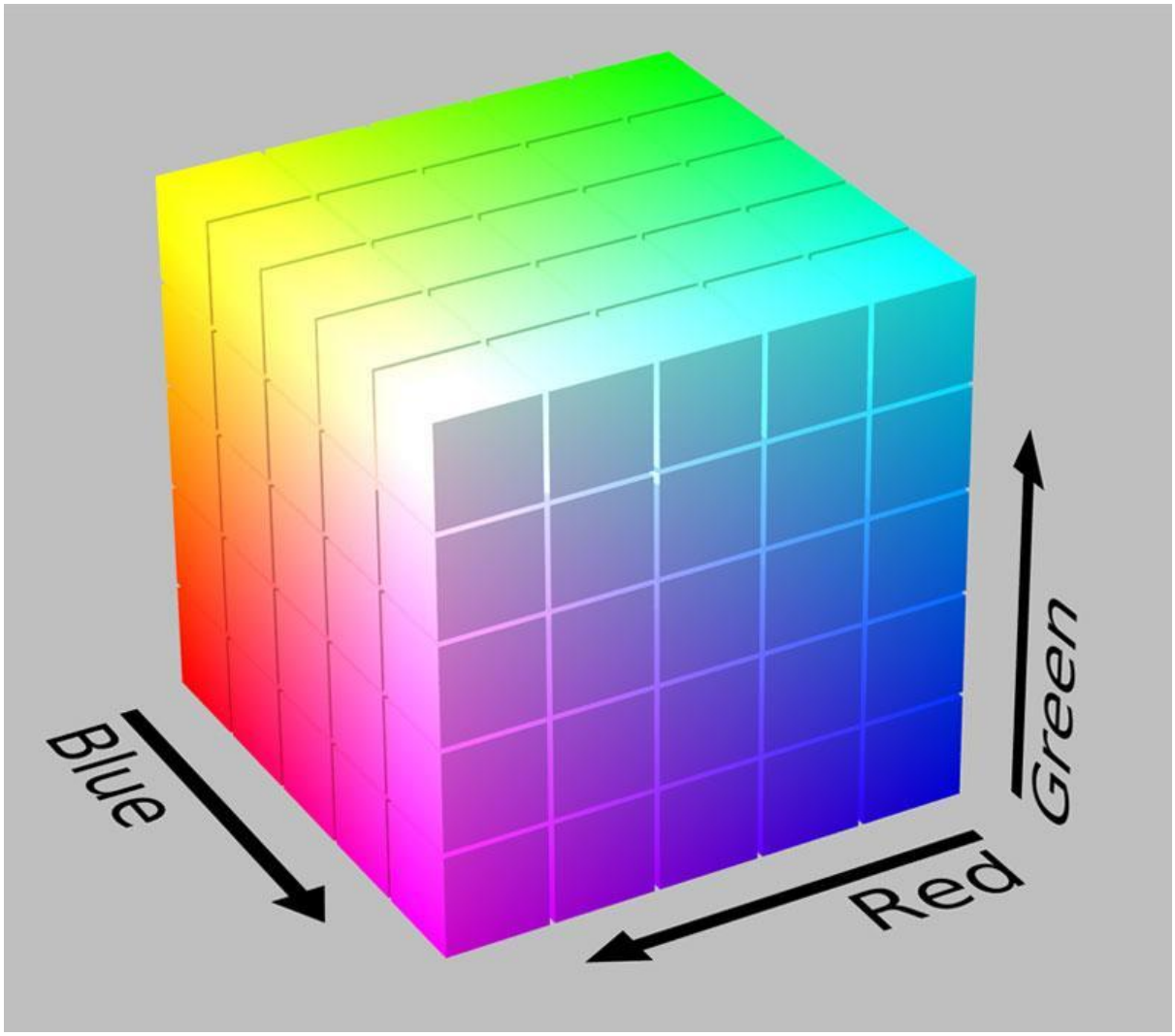
```
K(X, Y) = <X, Y> / (||X||*||Y||)
```

For more on cosine similarity refer wikipedia

HSV Preprocessing

Our image descriptor will be a 3D color histogram in the HSV color space (Hue, Saturation, Value). Typically, images are represented as a 3-tuple of Red, Green, and Blue (RGB). We often think of the RGB color space as a “cube”, as shown

below:



Why use HSV?

R, G, B in RGB are all correlated to the color luminance(what we loosely call intensity),i.e., We cannot separate color information from luminance. HSV or Hue Saturation Value is used to separate image luminance from color information. This makes it easier when we are working on or need luminance of the image/frame

Image-based search engine using labeled data

Our previous model was performing well after HSV preprocessing. But it had some problems:-

1. It was able to show similar images to an input image by comparing their similarity using cosine similarity but our demand from the model is to get it to show the exact faces of the person whose image is input (for example if we give it the image of Dr. Abdul Kalam as input we want it to output all the other images of Dr. Abdul Kalam only).
2. The global average pooling layer was not trained since it was not fit on the model as the model was an unlabeled one

To resolve these problems we'll now move on to creating a model with labeled data. More precisely we'll be creating a face recognition model with tensorflow using CNNs and transfer learning

Face recognition using labeled data

Data

For this, we'll be using the pins face recognition dataset from Kaggle (https://www.kaggle.com/hereisburak/pins-face-recognition)

Splitting the data

We'll start by splitting the data(containing 150+ images for each of 105 classes) into three folder (train,test,validation) using the splitfolders method(https://pypi.org/project/split-folders/) in the ratio- 0.8-0.15-0.05 (train,val,test)

Preprocessing data

To preprocess our data we will convert it into batches so our CPU/GPU is not overloaded. To convert it into batches we'll use the tf.keras.utils.image_dataset_from_directory (https://www.tensorflow.org/api_docs/python/tf/keras/utils/image_dataset_from_directory).

Model :

Feature extractor model

We'll start by building a feature extractor model using transfer learning same as above but this time we'll add the output layer as well and fit the model with our data.

We have used InceptionV3 in our case (https://paperswithcode.com/sota/image-classification-on-dtd)

(https://www.codeproject.com/Articles/5275262/AI-Face-Recognition-with-a-Pre-Trained-Model)

Layers in the model:

The model contains of 4 layers:

1. INPUT LAYER

This layer takes the images in our dataset as input having the shape (224,224,3)

2. FUNCTIONAL LAYER

This is the main layer of the model which contains the state of the art CNN model which we take from tf.keras.applications. For example:- EfficientNetB0,ResNet50,VGG16,etc. This layer will learn the features in the images necessary for classification. We'll build a feature-extractor transfer learning model from this which will learn patterns in our image data.

3. GLOBAL_AVERAGE_POOL LAYER

This layer will pool the output from the FUNCTIONAl LAYER and give us our pooled feature vector.

4. OUTPUT LAYER

This will be a dense layer which will have the same number of neurons or hidden units as that of the categories in the data. It will use a softmax activation function to output prediction probabilities of the different (in our case 105) categories.

After fitting the model on the dataset using categorical_crossentropy as the loss function and Adam optimizer the results we get our as follows:

- training accuracy: 0.3396
- validation accuracy: 0.2988
- test accuracy: 0.3017

Now to get better accuracy on our validation and testing set we'll move on to fine-tuning the model

Fine-tuning :

Fine-tuning is done after fitting the model for a few epochs on feature extractor model. In fine-tuning we turn the layers of the transfer learning model to trainable from non-trainable. In the feature extractor model the data learns the weights pre-trained in the transfer learning model while in fine-tuned it is trained on the spot and

learns weights according to our dataset to give better results on our particular dataset.

After compiling the model with all layers true and fitting the model the results are as follows:

- training accuracy: 0.8772
- validation accuracy: 0.7203
- test accuracy: 0.7197

We can see the results are much better after fine tuning

Let's test the model on our custom images next....

Face Detection and Feature Extraction

Our model's aim is to be able to detect multiple faces and extract their features from Image to be able to find similar Face images from and Image with multiple people as well.

Firstly, to detect multiple face we use Dlib's face detector. We detect faces, draw a rectangle around them and cut and save them to be able to use them for facial feature extraction

The code for the same is given below:

```
import cv2
from face_recognition_models import face_recognition_model_location
import dlib
import numpy as np
import face_recognition
import glob
import os
import pickle
from deepface import DeepFace
import tensorflow
from arcface import ArcFace

face_rec = ArcFace.ArcFace()

detector = dlib.get_frontal_face_detector()

def MyRec(rgb,x,y,w,h,v=25,color=(200,0,0),thikness =2):
    """To draw stylish rectangle around the objects"""
    cv2.line(rgb, (x,y),(x+v,y), color, thikness)
    cv2.line(rgb, (x,y),(x,y+v), color, thikness)

    cv2.line(rgb, (x+w,y),(x+w-v,y), color, thikness)
    cv2.line(rgb, (x+w,y),(x+w,y+v), color, thikness)

    cv2.line(rgb, (x,y+h),(x,y+h-v), color, thikness)
    cv2.line(rgb, (x,y+h),(x+v,y+h), color, thikness)

    cv2.line(rgb, (x+w,y+h),(x+w,y+h-v), color, thikness)
    cv2.line(rgb, (x+w,y+h),(x+w-v,y+h), color, thikness)

def save(img,name, bbox, width=180,height=227):
    x, y, w, h = bbox
```

```
    imgCrop = img[y:h, x: w]
    imgCrop = cv2.resize(imgCrop, (width, height))#we need this line to
reshape the images
    cv2.imwrite(name+".jpg", imgCrop)


# Define a function that loads and encodes an image
def load_and_save_img(filename,counter,path):

    # Load the input image
    frame =cv2.imread(filename)
    image =cv2.cvtColor(frame,cv2.COLOR_BGR2RGB)
    faces = detector(image)
    fit =20
    # detect the face
    #if len(faces)>1:

    for counter1,face in enumerate(faces):
        print(counter)
        x1, y1 = face.left(), face.top()
        x2, y2 = face.right(), face.bottom()
        cv2.rectangle(frame,(x1,y1),(x2,y2),(220,255,220),1)
        MyRec(frame, x1, y1, x2 - x1, y2 - y1, 1, (0,250,0), 3)
        #save(image,new_path+str(counter),(x1-fit,y1-fit,x2+fit,y2+fit))
        new_path = f'{path}_{counter}_{counter1}'
        save(frame,new_path,(x1,y1,x2,y2))
        #frame = cv2.resize(frame,(800,800))
        # cv2.imshow('img',frame)
        # cv2.waitKey(0)
        print("done saving")
```

After this we move on to creating a list of facial features of all our cropped face images.

For this we use VGG-Face architecture from DeepFace's facial feature extractor method "represent". We also Tried ArcFace, Dlib, EfficientNet for facial feature extraction but the best results were obtained from VGG-Face.

The full code with face detection and extraction is given below. We also saved the the feature vector list of all the images in a separate file for later use for testing and Comparision with test images through cosine similarity

```
import cv2
from face_recognition_models import face_recognition_model_location
import dlib
import numpy as np
import face_recognition
import glob
import os
import pickle
from deepface import DeepFace
import tensorflow
from arcface import ArcFace

face_rec = ArcFace.ArcFace()

detector = dlib.get_frontal_face_detector()

def MyRec(rgb,x,y,w,h,v=25,color=(200,0,0),thikness =2):
    """To draw stylish rectangle around the objects"""
    cv2.line(rgb, (x,y),(x+v,y), color, thikness)
    cv2.line(rgb, (x,y),(x,y+v), color, thikness)

    cv2.line(rgb, (x+w,y),(x+w-v,y), color, thikness)
    cv2.line(rgb, (x+w,y),(x+w,y+v), color, thikness)

    cv2.line(rgb, (x,y+h),(x,y+h-v), color, thikness)
    cv2.line(rgb, (x,y+h),(x+v,y+h), color, thikness)

    cv2.line(rgb, (x+w,y+h),(x+w,y+h-v), color, thikness)
    cv2.line(rgb, (x+w,y+h),(x+w-v,y+h), color, thikness)

def save(img,name, bbox, width=180,height=227):
    x, y, w, h = bbox
    imgCrop = img[y:h, x: w]
    imgCrop = cv2.resize(imgCrop, (width, height))#we need this line to
reshape the images
    cv2.imwrite(name+".jpg", imgCrop)


# Define a function that loads and encodes an image
def load_and_save_img(filename,counter,path):

    # Load the input image
    frame =cv2.imread(filename)
    image =cv2.cvtColor(frame,cv2.COLOR_BGR2RGB)
    faces = detector(image)
    fit =20
    # detect the face
    #if len(faces)>1:
```

```
    for counter1,face in enumerate(faces):
        print(counter)
        x1, y1 = face.left(), face.top()
        x2, y2 = face.right(), face.bottom()
        cv2.rectangle(frame,(x1,y1),(x2,y2),(220,255,220),1)
        MyRec(frame, x1, y1, x2 - x1, y2 - y1, 1, (0,250,0), 3)
        #save(image,new_path+str(counter),(x1-fit,y1-fit,x2+fit,y2+fit))
        new_path = f'{path}_{counter}_{counter1}'
        save(frame,new_path,(x1,y1,x2,y2))
        #frame = cv2.resize(frame,(800,800))
        # cv2.imshow('img',frame)
        # cv2.waitKey(0)
        print("done saving")



def encode_image(img):
    #model = DeepFace.Facenet512.loadModel()
    encoded_image = DeepFace.represent(img)
    print(encoded_image)

    # print(encoded_image)
    return encoded_image

# Store the encoded vec in a dictionary
count = 0
counter = 9
encoded_vec = {}
encoded_vec_list = []
image_vec = {}

for image in glob.glob('images/Train' + '/' + '*.*'):
    print(image)
    #count = count + 1
    counter= counter + 1
    load_and_save_img(image,counter,'images/extracted_images/')
    frame =cv2.imread(image)
    img =cv2.cvtColor(frame,cv2.COLOR_BGR2RGB)
    faces = detector(img)
    print(len(faces))
    if len(faces)>1:
        for i in range(len(faces)):
            print('saved_twice')
            count = count+1
            image_vec[count] = image
    else:
      count = count + 1
      image_vec[count] = image
for img in glob.glob('images/extracted_images' + '/' + '*.*'):
```

```
    print(img)
    try:
      img_encoding = encode_image(img)
    except:
        continue
    encoded_vec_list.append(img_encoding)

print(encoded_vec_list)


file_name = "encoded_vec_list2.pkl"
open_file = open(file_name,"wb")
pickle.dump(encoded_vec_list,open_file)


for img_test in glob.glob('images\extracted_images' + '/' + '*.*'):
    os.remove(img_test)
```

After training and saving all the image vectors in “encoded_vec_list2”, we can move on to testing.

For testing we use the same process of face detection, cropping the face image and forming the feature vector for the same. Then we compare it with the feature vectors of all the face vecots saved in “encoded_vec_list2”. We use cosine similarity which is explained above to compare the distance between the faces and see how similar are they. Then we print the similar images with a threshold of 60% similar and above.

The code for testing is given below:

```
import pickle
import cv2
import dlib
import numpy as np
import face_recognition
import glob
import os

from tensorflow.python.keras.losses import cosine_similarity
from helper_functions import detector,load_and_save_img,encode_image
from deepface import DeepFace
from arcface import ArcFace

face_rec = ArcFace.ArcFace()

open_file = open('encoded_vec_list2.pkl',"rb")
encoded_vec_list_loaded = pickle.load(open_file)
open_file.close()
```

```
# Testing our encodings

#test_path = load_and_save_img('images\Train/apj.jpg',20)
test_path = 'images/Test/apj_test.jpg'
count_test = 0
counter_test = 9
encoded_vec_list_test = []
counter_test= counter_test + 1
frame_test =cv2.imread(test_path)
img_test =cv2.cvtColor(frame_test,cv2.COLOR_BGR2RGB)
faces_test = detector(img_test)
print(len(faces_test))
if len(faces_test)>1:
    results = []
    load_and_save_img(test_path,counter_test,'images/test_extracted_images/')
    for img_test in glob.glob('images/test_extracted_images' + '/' + '*.*'):
        results = []
        result_no = []
        img_encoded_test = DeepFace.represent(img_test)
        for encoding in encoded_vec_list_loaded:
            result = 100*(cosine_similarity(encoding,img_encoded_test))
            if result.numpy() < 30:
                results.append(True)
            else:
                results.append(False)
        print(results)
        image_test = cv2.imread(test_path)
        cv2.imshow('Image',image_test)
        cv2.waitKey(0)
        for i in range(len(encoded_vec_list_loaded)):
            if results[i] == True:
                image = cv2.imread(image_vec_loaded[i+1])
                cv2.imshow('Image',image)
                cv2.waitKey(0)

else:
    img_encoding_test = DeepFace.respresent(test_path)



# Compare
    results=[]
    for encoding in encoded_vec_list_loaded:
        result = 100*(cosine_similarity(encoding,img_encoding_test))
        if result.numpy() < -60:
            results.append(True)
        else:
            results.append(False)
```

```
    print(results)
    image_test = cv2.imread(test_path)
    cv2.imshow('Image',image_test)
    cv2.waitKey(0)



    for i in range(len(encoded_vec_list_loaded)):
        if results[i] == True:
            image = cv2.imread(image_vec_loaded[i+1])
            cv2.imshow('Image',image)
            cv2.waitKey(0)
for img_test in glob.glob('images/test_extracted_images' + '/' + '*.*'):
    os.remove(img_test)
```

DEPLOYMENT OF MODEL

There’re many ways to deploy a model like html css js django flask streamlit, But for our model we’ll be using flask for deployment.

- **FLASK:-**

  Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.

- **STEPS INVOLVED IN DEPLOYMENT**

1. After importing all the required libraries, we’ll load our model in
2. We’ll create an “@app.route” which will be our index page.

```
@app.route('/', methods=['GET','POST'])
def main():
    return render_template('index.html')
```

3. We’ll create an HTML form which will take images from the user.
4. We’ll create another “app.route” for the submit page.
5. This will contain our preprocessing function which will preprocess the image to be compatible with the model
6. Then we'll pass this image through the model and obtain its feature_vec
7. we’ll compare this feature_vec with all the feature_vecs of our image database
8. Then it’ll output the images similar to user’s image

DESICOC Image Retreival System

127.0.0.1:5000

DESIDOC IMAGE RETRIEVAL SYSTEM

<>

<>

Upload Your Image :

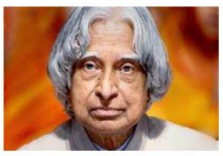
Choose File No file chosen

SUBMIT

TRAIN

Result:

Input Image:



Similar Images:








Flask Code:

```
from flask import Flask,render_template,request,flash,redirect
from deepface import DeepFace
import cv2
import deepface
import numpy as np
import glob
import os
import pickle
from deepface import DeepFace
import streamlit as st
import numpy as np
from PIL import Image

app = Flask(__name__)

@app.route('/', methods=['GET','POST'])
def main():
  return render_template('index.html')

@app.route('/submit', methods=['GET','POST'])
def index():
    if request.method == 'POST':
      # os.remove('images/ip/representations_vgg_face.pkl')
      imagefile = request.files['imagefile']
      img_path = 'image_data/' + imagefile.filename
      imagefile.save(img_path)
      verifications = DeepFace.find(img_path
,db_path='images/ip/',enforce_detection=False)
      paths = verifications['identity'].to_list()
      len_paths = int(0.5 * len(paths))
      paths = paths[:len_paths]
      print(paths)


    return render_template('index.html', img_path = paths)

@app.route('/train', methods=['GET','POST'])
def train():
    if request.method == 'POST':
      os.remove('images/ip/representations_vgg_face.pkl')
      imagefile = request.files['imagefile']
      img_path = 'image_data/' + imagefile.filename
      imagefile.save(img_path)
      verifications = DeepFace.find(img_path
,db_path='images/ip/',enforce_detection=False)
```

```
      paths = verifications['identity'].to_list()
      len_paths = int(0.5 * len(paths))
      paths = paths[:len_paths]
      print(paths)


    return render_template('index.html', img_path = paths)

if __name__ =='__main__':
  app.run(debug=False)
```

Conclusion:

With data becoming larger day by day due to social media and other platforms. Need for accurate predictions on unlabelled data is becoming need of the hour.

It is not possible to manually label a large set of images with their names and also not efficient. In this project we've built a web-app that takes input an Image and return back similar images meaning it tries to return the images of people having same/similar face to the input Image.

Our model can also find similar faces to that of the input face image from photos with multiple people using face detection.

Important Links:

https://pyimagesearch.com/

https://github.com/serengil/deepface

https://github.com/ageitgey/face_recognition