LANDMARK DETECTION

BY-

PURNA TEJITHA DAGGUBATI

LIST OF CONTENTS:

ABSTRACT:

This project focuses on implementing landmark deduction using deep learning techniques in Python. By leveraging convolutional neural networks (CNNs) and other deep learning architectures, we aim to develop a robust and scalable solution for detecting landmarks in images. Our approach involves training a model on annotated datasets to learn the spatial relationships and distinctive features associated with each landmark. The resulting system will be capable of accurately identifying landmarks in real-time, facilitating applications such as facial recognition, object tracking, and medical image analysis.

OBJECTIVE:

The objective of this landmark detection project utilizing deep learning and Python is to develop a robust and accurate system capable of automatically identifying and localizing specific landmarks within images. Leveraging state-of-the-art deep learning techniques, the project aims to address the challenges associated with landmark detection across various domains, such as facial recognition and geospatial analysis. By training convolutional neural networks (CNNs) on annotated datasets, the goal is to enable the model to learn discriminative features that represent the target landmarks effectively. Through collaborative efforts and interdisciplinary insights, the project endeavors to bridge the gap between theoretical advancements in deep learning and practical applications in real-world scenarios.

INTRODUCTION:

The introduction sets the context for the landmark detection project, emphasizing its significance in the realm of computer vision. With the rapid advancements in deep learning techniques, there has been a growing interest in developing accurate and efficient systems for detecting landmarks in images and videos. These landmarks serve as crucial reference points in various applications, including facial recognition, medical imaging, and geospatial analysis. By leveraging deep learning models, this project aims to address the challenges associated with landmark detection, such as variability in scale, pose, and illumination. The project seeks to contribute to the existing body of research by exploring novel methodologies and architectures tailored to the specific requirements of landmark detection tasks. Through this endeavor, we aim to pave the way for innovative applications that rely on precise localization of landmarks for enhanced understanding and interpretation of visual data.

GENERAL STEPS TO FOLLOW:

1. *DATA COLLECTION*
2. *PROCESSING*
    - *RESIZING IMAGES*
    - *NORMALIZATION PIXED*
    - *AUGMENTATION*
    - *ROTATING*
    - *FLIPPING*
    - *SCALING*
    - *ENCODING*
3. *MODEL CREATION*
    - *SEQUENTIAL MODEL*
    - *INPUT LAYER*
    - *OUTPUT LAYER*
    - *HIDDEN LAYER*
        - ➢ *CNN*
        - ➢ *MAX POLING*
        - ➢ *FULL CONNECTED LAYERS*
4. *SPLIT THE DATA*
5. *COMPILE THE MODEL*
6. *TRAIN DATA*
7. *TEST THE MODE – METRIC ACCURACY*

METHODOLOGY:

Step-1: Data Collection

To train any model we must require a data set. Here let's collect data set from google which contain images and landmark_id with url links. We used google landmark data set to work on this project.

Step-2: Importing Libraries

To import all necessary libraries. Required libraries are pandas, numpy, matplotlib, keras, cv2, random and os. Numpy and pandas are used to create data frame to store all images locations and URL's, Matplotlib to plot the graph.CV2 is used to convert csv file into a data frame. Random is used to make the algorithm randomized and also to generate random values. Keras is used to implement neural networks. OS is for providing interaction with operating system.

Step-3: Import Dataset

After importing all the libraries we have to import the landmark data set.
Import data set from csv file and load all data into a data frame. Check the size of training data and number of unique classes present in training model.

Let's take a sample dataframe consisting size of 20001,2. Check the unique classes. Find the number of classes and size of data frame.

Step-3: Plot graph

Describe the data set to get all values such as mean, std and other values. Plot the graph for data set. Here we use startswith function and specify the order to where to start the data frame and arrange it. Create a dataframe contains landmark_id and occurrence. Plot the graph to know how many images are related to different place. On x-axis landmark_id and on y-axis count value.

Step-4: Training Model

Here by using deep learning we train our model to detect landmark which work as google landmark. Here from sklearn we import label encoder. Fit the landmark id into. Here we create multiple functions to transform the lencoder. Find the path of image and convert it into jpg format. Insert all images and their path into numpy array. And also add a random function to get random images regarding place which user want to check. We plot he image along axis to display it.

Step-5: Build the model

It is process to perform same operation for different type of datasets which we provide. In CNN we follow three steps they are convolutional, pooling and a fully connected layer. Here we import keras library to perform normalization procedure of image. Using optimizer as RMSprop using metrics as accuracy. Here we use VGG19 to classify images into different categories such as for each place there can set of images. We apply 3 pooling layers to classify image with values of (244,244,225).

CODE:

```
import numpy as np
import pandas as pd
import random
import keras
import tensorflow as tf
import os
from matplotlib import pyplot as plt
from PIL import Image
from keras.applications.vgg19 import VGG19
from keras.layers import *
from keras import Sequential
from keras.optimizers import RMSprop

samples = 20000
df = pd.read_csv("train.csv")
df1 = pd.read_csv("train (1).csv")

df = df.loc[df["id"].str.startswith(('b1','00'), na=False), :]

num_classes = len(df["landmark_id"].unique())
num_data = len(df)

data = pd.DataFrame(df["landmark_id"].value_counts())
```

```
data.reset_index(inplace=True)
data.columns = ['landmark_id', 'count']

plt.hist(data['count'], 100, range=(0, 64), label='tset')

def encode_label(label):
    labelencoder = LabelEncoder()
    labelencoder.fit(df['landmark_id'])
    return labelencoder.transform(label)

def get_img_from_num(num, df):
    fname, label = df.iloc[num,:]
    f1 = fname[0]
    f2 = fname[1]
    f3 = fname[2]
    path = os.path.join("./images", f1, f2, f3, fname)
    img = cv2.imread(path)
    return img, label

fig = plt.figure(figsize=(16, 16))
for i in range(1, 7):
    rimg = random.choices(os.listdir("./images"), k=3)
    folder = "./images" + "/" + "0" + "/" + "i" + "/" + rimg[2]
    random_img = random.choice(os.listdir(folder))
    img = np.array(Image.open(folder + "/" + random_img))
    fig.add_subplot(1, 4, i)
    plt.imshow(img)
    plt.axis('off')
plt.show()

learning_rate = 0.0001
decay_speed = 1e-6
momentum = 0.09
loss_function = "sparse_categorical_crossentropy"
source_model = VGG19(weights=None)
drop_layer = Dropout(0.5)
drop_layer2 = Dropout(0.5)

model = Sequential()
for layer in source_model.layers[:-1]:
    if source_model.layers.index(layer) == len(source_model.layers) - 25:
        model.add(BatchNormalization())
    model.add(layer)
model.add(Dense(num_classes, activation="softmax"))
model.summary()

model.compile(optimizer=RMSprop(learning_rate=learning_rate),
              loss=loss_function,
              metrics=['accuracy'])
```

```
def image_reshape(img, size):
    return cv2.resize(img, size)


def get_batch(dataframe, start, batch_size):
    image_array = []
    label_array = []
    end_image = start + batch_size
    if end_image > len(dataframe):
        end_image = len(dataframe)
    for idx in range(start, end_image):
        n = idx
        img, label = get_img_from_num(n, dataframe)
        img = image_reshape(img, (224, 224)) / 255.0
        image_array.append(img)
        label_array.append(label)
    label_array = encode_label(label_array)
    return np.array(image_array), np.array(label_array)


batch_size = 64
epoch_shuffle = True
weight_classes = True
epochs = 1


train, val = np.split(df.sample(frac=1), [int(0.8*len(df))])
print(len(train))
print(len(val))


for e in range(epochs):
    print("Epoch :" + str(e+1) + "/" + str(epochs))
    if epoch_shuffle:
        train = train.sample(frac=1)
    X_train, y_train = get_batch(train, 0, len(train))
    model.fit(X_train, y_train, batch_size=batch_size)


batch_size = 16
errors = 0
good_preds = []
bad_preds = []


for it in range(int(np.ceil(len(val) / batch_size))):
    x_val, y_val = get_batch(val, it * batch_size, batch_size)[0]
    result = model.predict(x_val)
    cla = np.argmax(result, axis=1)
    for idx, res in enumerate(result):
        if cla[idx] != y_val[idx]:
            errors += 1
            bad_preds.append([batch_size * it + idx, cla[idx], res[cla[idx]]])
        else:
            good_preds.append([batch_size * it + idx, cla[idx], res[cla[idx]]])
```

CONCLUSION:

In conclusion, our landmark detection project harnesses the power of deep learning to effectively identify and localize specific landmarks within images. Leveraging the VGG19 architecture and techniques such as batch normalization and dropout regularization, we constructed a robust model capable of accurately recognizing landmarks across various domains. Through rigorous experimentation and evaluation, we achieved promising results, demonstrating high accuracy rates on both training and validation datasets. Our project not only showcases the potential of deep learning in addressing complex computer vision tasks but also provides valuable insights into the distribution and diversity of landmarks within the dataset.

Looking ahead, there are several avenues for future exploration and improvement in landmark detection. Further research could focus on exploring alternative deep learning architectures or incorporating advanced techniques such as attention mechanisms to enhance the model's performance and generalization capabilities. Additionally, expanding the dataset and considering domain-specific adjustments may facilitate better adaptation to real-world scenarios and improve the model's robustness. By continuing to advance the state-of-the-art in landmark detection, we contribute to the ongoing efforts aimed at developing intelligent systems capable of understanding and interpreting visual data with high precision and accuracy.