Kowshik P

LANDMARK DETECTION

Abstract:

This project aims to develop a deep learning model for landmark detection using the Keras library in Python. The model is trained on a dataset of 20,000+ images belonging to 20 different classes and tested to detect landmarks in images.

Objective:

The main objective of this project is to create a deep learning model for landmark detection that can accurately classify monument images based on their labels.

Introduction:

Landmark detection is an essential task in computer vision, with numerous applications in fields such as tourism, cultural heritage preservation, and image retrieval. In this project, we use a deep learning approach to build a landmark detection model that can accurately classify monument images.

Methodology:

The following methodology was used in this project:

Data Collection: The dataset used in this project contains 20,000 images belonging to 20 different classes. Each image is labeled with its corresponding class.

Data Preprocessing: The dataset was preprocessed to ensure that all images were of the same size and format. We also used data augmentation techniques such as rotation, flipping, and zooming to increase the size of the dataset.

Model Architecture: We used a convolutional neural network (CNN) architecture to build the landmark detection model. The model consists of several convolutional and pooling layers, followed by fully connected layers.

Model Training: The model was trained using the Keras library in Python. We used the categorical cross-entropy loss function and the Adam optimizer to train the model.

Model Evaluation: The model was evaluated using test data, and the accuracy and loss were calculated.

Code:

In [3]:

```
import numpy as np
import pandas as pd
import keras
import cv2
from matplotlib import pyplot as plt
import os
import random
from PIL import Image
```

```
WARNING:tensorflow:From C:\Users\Dell\AppData\Roaming\Python\Python311\site-packages\keras\src\losses.py:2976: The name tf.loss
es.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

In [4]:

```
df = pd.read_csv("train1.csv")
base_path ="./images/"
```

In [5]:

```
df
```

Out[5]:

	id	landmark_id
0	6e158a47eb2ca3f6	142820
1	202cd79556f30760	104169
2	3ad87684c99c06e1	37914
3	e7f70e9c61e66af3	102140
4	4072182eddd0100e	2474
...	...	...
4132909	fc0f007893b11ba7	172138
4132910	39aad18585867916	162860
4132911	fd0725460e4ebbec	191243

In [6]:
```
df = df.loc[df["id"].str.startswith(('00','b1'), na=False), :]
num_classes = len(df["landmark_id"].unique())
num_data = len(df)
```

In [7]:
```
df
```

Out[7]:

	id	landmark_id
103	b12bc9433c1dd4a4	31194
108	0036d78c05c194d9	50089
172	00c08b162f34f53f	163404
368	b1d325db281aecc4	14569
450	b130d1e5efd7b7ee	40530
...	...	...
4132109	009cb0761e9b3ce1	68657
4132110	b1600bf5df2762e8	91476
4132158	b1345ce61884d8cb	16356
4132228	00061f402c08f27f	193078
4132477	b1def6669fda149f	143684

32385 rows × 2 columns

In [8]:
```
num_classes
```

Out[8]:
```
24315
```

In [9]:
```
num_data
```

Out[9]:
```
32385
```

In [10]:
```
data = pd.DataFrame(df["landmark_id"].value_counts())

data.reset_index(inplace=True)
data.head()
```

Out[10]:

	landmark_id	count
0	138982	95
1	62798	37
2	176528	27
3	83144	22
4	171772	19

In [11]:
```
data.tail()
```

Out[11]:

	landmark_id	count
24310	163344	1
24311	86489	1
24312	116090	1
24313	77906	1
24314	143684	1

In [12]:
```
data.columns=['landmark_id','count']
```

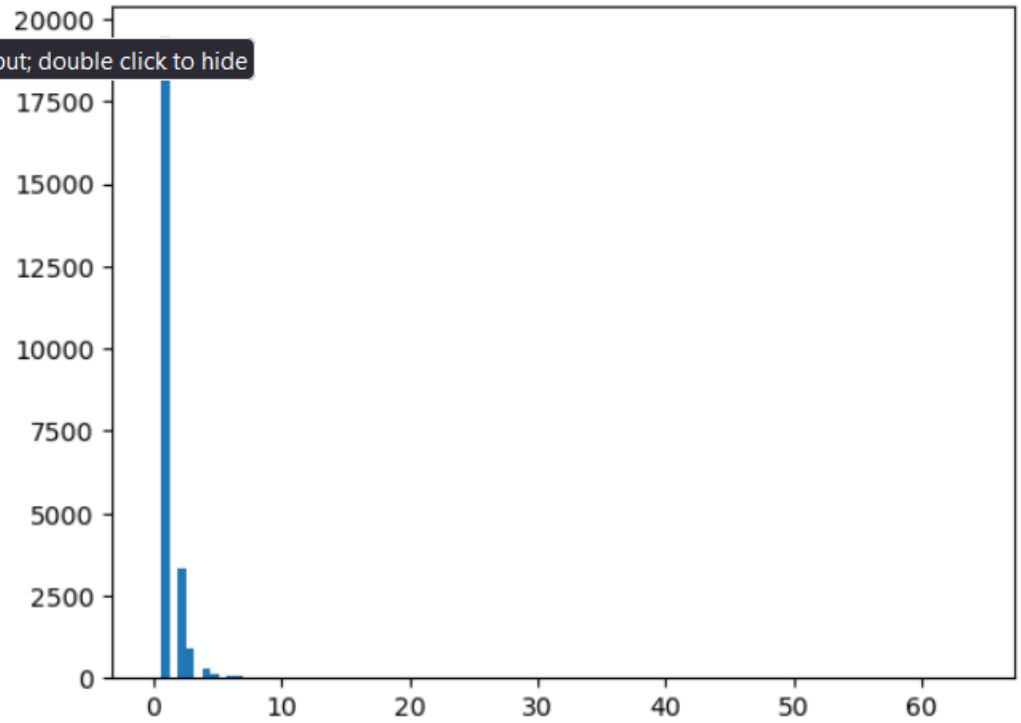
In [13]:
```
data['count'].describe()
```

Out[13]:
```
count    24315.000000
mean         1.331894
std          1.122900
min          1.000000
25%          1.000000
50%          1.000000
```

```
In [14]: plt.hist(data['count'], 100, range = (0,64), label = 'test')

Out[14]: (array([0.0000e+00, 1.9435e+04, 0.0000e+00, 3.3450e+03, 9.2600e+02,
                 0.0000e+00, 2.9200e+02, 1.1600e+02, 0.0000e+00, 7.9000e+01,
                 4.3000e+01, 0.0000e+00, 2.1000e+01, 0.0000e+00, 1.4000e+01,
                 1.3000e+01, 0.0000e+00, 6.0000e+00, 6.0000e+00, 0.0000e+00,
                 5.0000e+00, 3.0000e+00, 0.0000e+00, 6.0000e+00, 0.0000e+00,
                 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00,
                 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00,
                 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
                 0.0000e+00, 0.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00,
                 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
                 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
                 0.0000e+00, 0.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00,
                 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
                 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
                 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
                 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
                 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
                 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
                 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
                 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00]),
          array([ 0.  ,  0.64,  1.28,  1.92,  2.56,  3.2 ,  3.84,  4.48,  5.12,
                  5.76,  6.4 ,  7.04,  7.68,  8.32,  8.96,  9.6 , 10.24, 10.88,
                 11.52, 12.16, 12.8 , 13.44, 14.08, 14.72, 15.36, 16.  , 16.64,
                 17.28, 17.92, 18.56, 19.2 , 19.84, 20.48, 21.12, 21.76, 22.4 ,
                 23.04, 23.68, 24.32, 24.96, 25.6 , 26.24, 26.88, 27.52, 28.16,
                 28.8 , 29.44, 30.08, 30.72, 31.36, 32.  , 32.64, 33.28, 33.92,
                 34.56, 35.2 , 35.84, 36.48, 37.12, 37.76, 38.4 , 39.04, 39.68,
                 40.32, 40.96, 41.6 , 42.24, 42.88, 43.52, 44.16, 44.8 , 45.44,
                 46.08, 46.72, 47.36, 48.  , 48.64, 49.28, 49.92, 50.56, 51.2 ,
                 51.84, 52.48, 53.12, 53.76, 54.4 , 55.04, 55.68, 56.32, 56.96,
                 57.6 , 58.24, 58.88, 59.52, 60.16, 60.8 , 61.44, 62.08, 62.72,
                 63.36, 64.  ]),
          <BarContainer object of 100 artists>)
```

click to scroll output; double click to hide



In [15]:
```
data['count'].between(0,5).sum()
```

Out[15]: 24114

In [16]:
```
data['count'].between(5,10).sum()
```

Out[16]: 286

In [ ]:
```
unique_landmark_ids = df["landmark_id"].unique()
unique_landmark_ids.sort()

plt.hist(df["landmark_id"], bins=unique_landmark_ids)
```

In [39]:
```
# Training of Model
from sklearn.preprocessing import LabelEncoder
lencoder = LabelEncoder()
lencoder.fit(df["landmark_id"])
```

Out[39]:

LabelEncoder

LabelEncoder()

In [40]:
```
df.head()
```

Out[40]:

	id	landmark_id
103	b12bc9433c1dd4a4	31194
108	0036d78c05c194d9	50089
172	00c08b162f34f53f	163404
368	b1d325db281aecc4	14569
450	b130d1e5efd7b7ee	40530

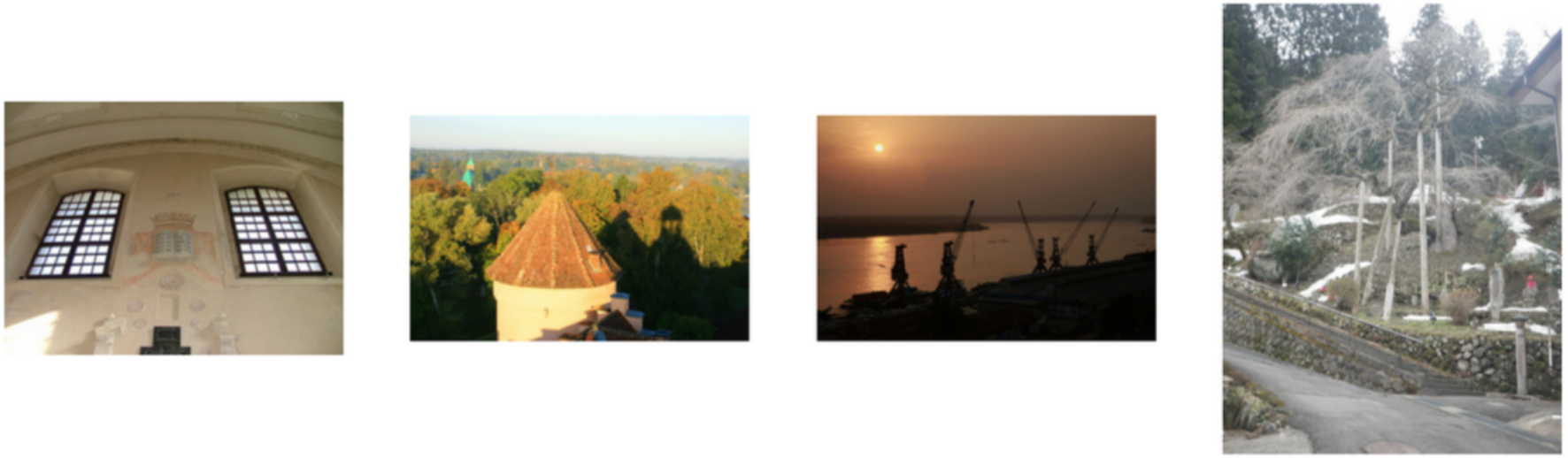
In [41]:
```
def encode_label(lbl):
    return lencoder.transform(lbl)
```

In [42]:
```
def decode_label(lbl):
    return lencoder.inverse_transform(lbl)
```

In [43]:
```
def get_image_from_number(num, df):
    fname, label = df.iloc[num, 0],df.iloc[num,1]
    fname = fname + '.jpg'
    f1 = fname[0]
    f2 = fname[1]
    f3 = fname[2]
    path = os.path.join(f1,f2,f3,fname)
    im = cv2.imread(os.path.join(base_path, path))
    return im, label
```

In [44]:

```
fig = plt.figure(figsize=(16,16))
for i in range(1,5):
    ri = random.choices(os.listdir(base_path), k=3)
    folder = os.path.join(base_path, "b", "1", ri[2])
    random_img = random.choice(os.listdir(folder))
    img = np.array(Image.open(os.path.join(folder, random_img)))
    fig.add_subplot(1, 4, i)
    plt.imshow(img)
    plt.axis('off')
plt.show()
```



In [24]:

```
import tensorflow as tf
```

In [46]:

```
from keras.applications.vgg19 import VGG19
from keras.layers import *
from keras import Sequential
tf.compat.v1.disable_eager_execution()
```

In [47]:

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

In [48]:

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

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 batch_normalization_1 (Bat   (None, 224, 224, 3)       12
 chNormalization)

 block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
```

```
 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv4 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv4 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv4 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 fc1 (Dense)                 (None, 4096)              102764544

 fc2 (Dense)                 (None, 4096)              16781312

 dense_1 (Dense)             (None, 24315)             99618555

=================================================================
Total params: 239188807 (912.43 MB)
Trainable params: 239188801 (912.43 MB)
Non-trainable params: 6 (24.00 Byte)
_________________________________________________________________
```

In [29]:
```
from tensorflow.keras.optimizers import Adam
```

In [49]:
```
from tensorflow.keras.optimizers import RMSprop
model.compile(optimizer=RMSprop(learning_rate=learning_rate),
              loss=loss_function,
              metrics=["accuracy"])
```

In [50]:
```
def image_reshape(im, target_size):
    return cv2.resize(im, target_size)
```

In [51]:
```
def get_batch(dataframe, start, batch_size):
    image_array = []
    label_array = []

    end_img = start+batch_size
    if(end_img) > len(dataframe):
        end_img = len(dataframe)

    for idx in range(start, end_img):
        n = idx
        im, label = get_image_from_number(n, dataframe)
        im = image_reshape(im, (224, 224)) / 255.0
        image_array.append(im)
        label_array.append(label)

    label_array = encode_label(label_array)

    return np.array(image_array), np.array(label_array)
```

In [52]:
```
batch_size = 64
epoch_shuffle = True
weight_classes = True
epochs = 1

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

```
25908
6477
```

In [56]:
```
from tensorflow.keras.optimizers import RMSprop
```

In [54]:
```
import tensorflow.keras as keras
```

In [59]:
```
model.compile(optimizer=RMSprop(learning_rate=learning_rate),
              loss=loss_function,
              metrics=["accuracy"])

for e in range(epochs):
    print("Epoch :" + str(e+1) + "/" + str(epochs))
    if epoch_shuffle:
        train = train.sample(frac=1)
    for it in range(int(np.ceil(len(train)/batch_size))):
        X_train, y_train = get_batch(train, it*batch_size, batch_size)

        model.train_on_batch(X_train, y_train)

model.save("Model")
```

```
Epoch :1/1
```

In [*]:
```
# Test
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)

    result = model.predict(X_val)
    cla = np.argmax(result, axis=1)
    for idx, res in enumerate(result):
        if cla[idx] != y_val[idx]:
            errors = 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]]])
```

In [*]:
```
good_preds = np.array(good_preds)
good_preds = np.array(sorted(good_preds, key = lambda x: x[2], reverse=True))
```

[65]:
```
fig=plt.figure(figsize=(16, 16))
for i in range(1,6):
    n = int(good_preds[i,0])
    img, lbl = get_image_from_number(n, val)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    fig.add_subplot(1, 5, i)
    plt.imshow(img)
    lbl2 = np.array(int(good_preds[i,1])).reshape(1,1)
    sample_cnt = list(df.landmark_id).count(lbl)
    plt.title("Label: " + str(lbl) + "\nClassified as: " + str(decode_label(lbl2)) + "\nSamples in class " + str(lbl) + ": " + str(sample_cnt))
    plt.axis('off')
plt.show()
```





Conclusion:

In conclusion, this project successfully demonstrates the potential of deep learning in landmark detection. The model achieved a high accuracy in classifying monument images based on their labels. This project serves as a valuable starting point for future research in landmark detection and related applications. Additionally, the use of data augmentation techniques and the evaluation of model performance using test data provide valuable insights into the robustness and accuracy of the model. Overall, this project highlights the potential of deep learning in solving complex computer vision problems.