Landmark Detection

-By M.Kaushik Sai

ABSTRACT:

Landmark detection is a pivotal task within the realm of computer vision, serving as a cornerstone for a multitude of applications ranging from tourism and navigation to augmented reality experiences. The primary goal of this project is to devise an efficient and accurate methodology for identifying landmarks depicted in images. This objective is pursued through the utilization of advanced deep learning techniques, which aim to classify images into distinct landmark categories.

In this project, we utilize a pre-trained VGG19 CNN architecture as the foundation for landmark detection. Through fine-tuning and augmentation techniques, we seek to enhance the model's ability to generalize to unseen data while mitigating issues such as overfitting. Furthermore, the utilization of modern optimization algorithms and data preprocessing techniques further contributes to the robustness and efficiency of the proposed methodology.

By implementing and evaluating this methodology, we can understand its efficacy in accurately identifying landmarks from images. Through experimentation and analysis, we seek to validate the performance and applicability of our approach across diverse datasets and real-world scenarios. Ultimately, this project endeavors to understand and develop landmark detection mechanism further by using relevant architecture , datasets(taken from google).

OBJECTIVE:

The primary objective of this project is to develop a robust model capable of accurately identifying landmarks from images. By leveraging deep learning algorithms and techniques, we aim to achieve classification of landmarks even in the presence of challenges such as variations in viewpoint, surrounding environment ,etc.

INTRODUCTION:

Landmark detection plays a vital role in many real-world applications, including image retrieval, tourism recommendation systems, and cultural heritage preservation. Traditional methods for landmark detection often rely on handcrafted features and shallow learning algorithms, which may struggle with the complexity and variability of real-world data.

In recent years, deep learning has emerged as a powerful approach for image classification and object detection tasks. Convolutional Neural Networks (CNNs), in particular, have shown remarkable performance in various computer vision tasks, including landmark detection. By learning hierarchical features directly from raw pixel data, CNNs can capture intricate patterns and variations present in images using kernels /filters making them well-suited for landmark detection.

- In this project, we propose a methodology for landmark detection using a pre-trained VGG19 CNN architecture. We leverage transfer learning by fine-tuning the VGG19 model on a dataset of landmark images which has the following:
- **Input Layer:** Receives input images, typically with a size of 224x224 pixels.
- **Convolutional Layers:** Consists of 16 layers, each applying a 3x3 filter to extract features, followed by a ReLU activation function.
- **Pooling Layers:** After every two convolutional layers, max-pooling layers with a 2x2 filter and a stride of 2 downsample the feature maps.
- **Fully Connected Layers:** Three fully connected layers follow the convolutional and pooling layers, performing high-level feature extraction.
- **Output Layer:** The final fully connected layer, followed by a softmax activation function, outputs the predicted probabilities for each class in a classification task.

In essence, VGG19 uses a stack of convolutional layers followed by pooling layers for feature extraction, topped with fully connected layers for classification, making it effective for various computer vision tasks.

METHODOLOGY:

1. **Data Preprocessing:** We begin by preprocessing the dataset, which includes loading the images, resizing them to a standard size, and normalizing pixel values.It is hard to work with raw categorial data , instead we encode the landmark labels using a LabelEncoder to convert them into numerical representations .
2. **Model Architecture:** As discussed previously, we utilize a pre-trained VGG19 model as the backbone architecture for landmark detection. We remove the final classification layer of VGG19 and append a new fully connected layer followed by a softmax activation function to predict landmark classes. Batch normalization is applied to stabilize and accelerate the training process.
3. **Training:** The model is trained using the RMSprop optimizer with a specified learning rate and momentum. We employ a mini-batch stochastic gradient descent approach, where training samples are processed in batches to improve convergence and efficiency. During training, we monitor the loss function and accuracy metrics to assess model performance. The time taken for training varies on number of epochs implemented.
4. **Evaluation:** It is necessary to understand how well the model is performing so as to further fine tune any parameters. After training, the model is evaluated on a separate validation dataset to assess its generalization ability. We compute metrics such as accuracy and error rate to measure the model's performance in classifying unseen images.

CODE: The provided Python code implements the proposed methodology for landmark detection. It includes data loading, preprocessing, model definition, training, and evaluation stages. Key libraries such as TensorFlow, Keras, OpenCV, and NumPy are utilized for implementation. The code is structured into modular functions for clarity and reusability.

For better understanding I have used Jupiter notebook and have taken screenshots.

```
Landmark Detection
```

markdown

For the above project we import the following modules

pandas as pd: Used for data manipulation and analysis, especially for handling tabular data (e.g., CSV files).

matplotlib.pyplot as plt: Used for creating visualizations such as histograms, line plots, scatter plots, etc.

sklearn.preprocessing.LabelEncoder: Used to encode categorical labels into numerical values for machine learning tasks.

cv2: OpenCV library for computer vision tasks like reading and manipulating images.

PIL.Image: Python Imaging Library for opening, manipulating, and saving various image file formats.

os: Provides functions for interacting with the operating system, like working with file paths and directories.

random: Generates random numbers and selections, useful for adding randomness to data processing tasks.

numpy as np: Fundamental package for scientific computing with Python, especially for numerical operations on arrays and matrices.

```python
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.preprocessing import LabelEncoder
import cv2
from PIL import Image
import os
import random
import numpy as np
```

[1] Python

```
c:\Users\tanvi\anaconda3\lib\site-packages\scipy\__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.26.3
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

Read the CSV file into a DataFrame

```python
df=pd.read_csv("D:\\landmark_det\\train.csv")
```

[2] Python

```python
df
```

[3] Python

	id	landmark_id
0	17660ef415d37059	1
1	92b6290d571448f6	1
2	cd41bf948edc0340	1
3	fb09f1e98c6d2f70	1
4	25c9dfc7ea69838d	7
...	...	...
1580465	72c3b1c367e3d559	203092
1580466	7a6a2d9ea92684a6	203092
1580467	9401fad4c497e1f9	203092
1580468	aacc960c9a228b5f	203092
1580469	d9e338c530dca106	203092

1580470 rows × 2 columns

Filter DataFrame to include only rows where 'id' starts with '00'

```python
samples = 20000
df = df.loc[df["id"].str.startswith('00', na=False), :]
num_classes = len(df["landmark_id"].unique())
num_data = len(df)
```

[4] Python

Number of unique landmark IDs in this subset.

Number of unique landmark IDs in this subset.

+ Code + Markdown

| 💡 Click here to ask Blackbox to help you code faster |

```
num_classes
```

Python

```
5346
```

Total number of rows in the DataFrame

| 💡 Click here to ask Blackbox to help you code faster |

```
num_data
```

Python

```
6120
```

Create a DataFrame to store the count of each landmark ID

| 💡 Click here to ask Blackbox to help you code faster |

```
data=pd.DataFrame(df["landmark_id"].value_counts())
#Reset the index of the 'data' DataFrame
data.reset_index(inplace=True)

data.head()
```

[7] Python

	index	landmark_id
0	138982	31
1	83144	14
2	126637	7
3	194914	7
4	109169	6

head displays first few rows and tail displays last few rows

| 💡 Click here to ask Blackbox to help you code faster |

```
data.tail()
```

[8] Python

	index	landmark_id
5341	71434	1
5342	71336	1
5343	71228	1
5344	71145	1
5345	202981	1

We rename the columns

We rename the columns

| 💡 Click here to ask Blackbox to help you code faster |

```
data.columns=["landmark_id","count"]
```

Python

| 💡 Click here to ask Blackbox to help you code faster |

```
data['count'].describe()
```

Python

```
count    5346.000000
mean        1.144781
std         0.641260
min         1.000000
25%         1.000000
50%         1.000000
75%         1.000000
max        31.000000
Name: count, dtype: float64
```

For visualization we plot histograms of landmark ID counts with different ranges

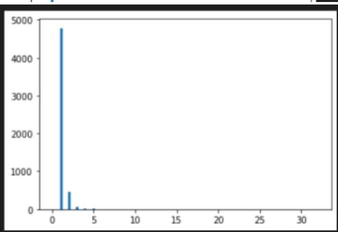
| 💡 Click here to ask Blackbox to help you code faster |

```
plt.hist(data['count'],bins=100,range=(0,32),label="test")
```

Python

```
(array([0.000e+00, 0.000e+00, 0.000e+00, 4.781e+03, 0.000e+00, 0.000e+00,
        4.520e+02, 0.000e+00, 0.000e+00, 7.500e+01, 0.000e+00, 0.000e+00,
        2.200e+01, 0.000e+00, 0.000e+00, 9.000e+00, 0.000e+00, 0.000e+00,
```

```
(array([0.000e+00, 0.000e+00, 0.000e+00, 4.781e+03, 0.000e+00, 0.000e+00,
        4.520e+02, 0.000e+00, 0.000e+00, 7.500e+01, 0.000e+00, 0.000e+00,
        2.200e+01, 0.000e+00, 0.000e+00, 9.000e+00, 0.000e+00, 0.000e+00,
        3.000e+00, 0.000e+00, 0.000e+00, 2.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 1.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        1.000e+00, 0.000e+00, 0.000e+00, 0.000e+00]),
 array([ 0.  ,  0.32,  0.64,  0.96,  1.28,  1.6 ,  1.92,  2.24,  2.56,
         2.88,  3.2 ,  3.52,  3.84,  4.16,  4.48,  4.8 ,  5.12,  5.44,
         5.76,  6.08,  6.4 ,  6.72,  7.04,  7.36,  7.68,  8.  ,  8.32,
         8.64,  8.96,  9.28,  9.6 ,  9.92, 10.24, 10.56, 10.88, 11.2 ,
        11.52, 11.84, 12.16, 12.48, 12.8 , 13.12, 13.44, 13.76, 14.08,
        14.4 , 14.72, 15.04, 15.36, 15.68, 16.  , 16.32, 16.64, 16.96,
        17.28, 17.6 , 17.92, 18.24, 18.56, 18.88, 19.2 , 19.52, 19.84,
        20.16, 20.48, 20.8 , 21.12, 21.44, 21.76, 22.08, 22.4 , 22.72,
        23.04, 23.36, 23.68, 24.  , 24.32, 24.64, 24.96, 25.28, 25.6 ,
        25.92, 26.24, 26.56, 26.88, 27.2 , 27.52, 27.84, 28.16, 28.48,
        28.8 , 29.12, 29.44, 29.76, 30.08, 30.4 , 30.72, 31.04, 31.36,
        31.68, 32.  ]),
 <BarContainer object of 100 artists>)
```

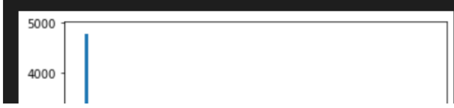


| 💡 Click here to ask Blackbox to help you code faster |

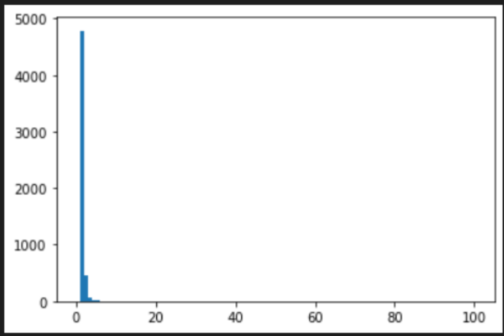
```
plt.hist(data['count'],bins=100,range=(0,100),label="changes")#did not run this
```

```
(array([0.000e+00, 4.781e+03, 4.520e+02, 7.500e+01, 2.200e+01, 9.000e+00,
        3.000e+00, 2.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 1.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 1.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
```

```
(array([0.000e+00, 4.781e+03, 4.520e+02, 7.500e+01, 2.200e+01, 9.000e+00,
        3.000e+00, 2.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 1.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 1.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00]),
 array([  0.,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  10.,
         11.,  12.,  13.,  14.,  15.,  16.,  17.,  18.,  19.,  20.,  21.,
         22.,  23.,  24.,  25.,  26.,  27.,  28.,  29.,  30.,  31.,  32.,
         33.,  34.,  35.,  36.,  37.,  38.,  39.,  40.,  41.,  42.,  43.,
         44.,  45.,  46.,  47.,  48.,  49.,  50.,  51.,  52.,  53.,  54.,
         55.,  56.,  57.,  58.,  59.,  60.,  61.,  62.,  63.,  64.,  65.,
         66.,  67.,  68.,  69.,  70.,  71.,  72.,  73.,  74.,  75.,  76.,
         77.,  78.,  79.,  80.,  81.,  82.,  83.,  84.,  85.,  86.,  87.,
         88.,  89.,  90.,  91.,  92.,  93.,  94.,  95.,  96.,  97.,  98.,
         99., 100.]),
 <BarContainer object of 100 artists>)
```



```
       88.,  89.,  90.,  91.,  92.,  93.,  94.,  95.,  96.,  97.,  98.,
       99., 100.]),
 <BarContainer object of 100 artists>)
```



We Count the number of classes with between 0 and 5 samples

| 💡 Click here to ask Blackbox to help you code faster |

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

```
5339
```

| 💡 Click here to ask Blackbox to help you code faster |

```
data
```

| 💡 Click here to ask Blackbox to help you code faster |

```
data
```

[15]

...

	landmark_id	count
0	138982	31
1	83144	14
2	126637	7
3	194914	7
4	109169	6
...	...	...
5341	71434	1
5342	71336	1
5343	71228	1
5344	71145	1
5345	202981	1

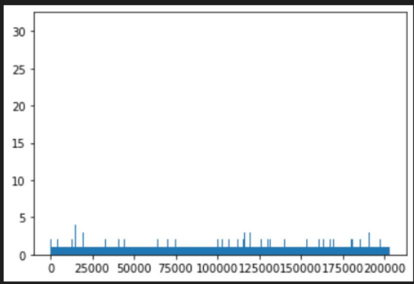
5346 rows × 2 columns

Now we plot another histogram of landmark IDs for intervals of unique landmark ids

| 💡 Click here to ask Blackbox to help you code faster |

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

```
(array([2., 1., 1., ..., 1., 1., 2.]),
 array([    27,     60,    124, ..., 202950, 202972, 202981], dtype=int64),
 <BarContainer object of 5345 artists>)
```



Initialize a LabelEncoder object to convert categorial data into numerical values.

+ Code + Markdown

| 💡 Click here to ask Blackbox to help you code faster |

```
lencoder=LabelEncoder()
lencoder.fit(df['landmark_id'])
```

```
▾ LabelEncoder
LabelEncoder()
```

| 💡 Click here to ask Blackbox to help you code faster |

```
df.head()
```

	id	landmark_id
119	00cba0067c078490	27
120	00f928e383e1d121	27
796	009ecdb56b5e9adb	60
1089	00d5d47528839144	124
1133	00e9003a381ab809	134

We make a function to encode labels using the fitted LabelEncoder

Comment Code | | 💡 Click here to ask Blackbox to help you code faster | Comment Code |

```
def encode_label(label):
    return lencoder.transform(label)
```

Like wise we make a function to decode encoded labels using the fitted LabelEncoder

Like wise we make a function to decode encoded labels using the fitted LabelEncoder

```
Comment Code | | 💡 Click here to ask Blackbox to help you code faster | Comment Code |
def decode_label(label):
    return lencoder.inverse_transform(label)
```

To access the images I Set the base path for image files and a function to retrieve an image and its label from the DataFrame using its index

```
| 💡 Click here to ask Blackbox to help you code faster |
base_path='D:\\Images\\'
Comment Code | Comment Code
def get_image_from_number(num, df):
    fname, label = df.iloc[num, :]
    fname = fname + '.jpg'
    image_name=fname
    for root, dirs, files in os.walk(base_path):
        if image_name in files:
            path =os.path.join(root, image_name)
    im = cv2.imread(path)
    return im, label
```

To Display random images from random subdirectories I used for loop to go through each directory and then randomly select an image.

```
| 💡 Click here to ask Blackbox to help you code faster |
base_path='D:/Images/'
# Create a figure to display random images
fig = plt.figure(figsize=(16,16))
# Get a list of subdirectories in the base path
a=os.listdir(base_path)
for i in range(1,5):
    folder=base_path+random.choice(a)+"/"+random.choice(["0/0/"])
    print(folder)
    folderf=folder+random.choice(os.listdir(folder))
    print(folderf)
    random_img = random.choice(os.listdir(folderf))
    img = np.array(Image.open(folderf+ '/'+random_img))
    fig.add_subplot(1,4,i)
    plt.imshow(img)
    plt.axis('off')
plt.show()
```

```
D:/Images/image0/0/0/
D:/Images/image0/0/0/7
D:/Images/image1/0/0/
D:/Images/image1/0/0/d
D:/Images/image1/0/0/
D:/Images/image1/0/0/d
D:/Images/image0/0/0/
D:/Images/image0/0/0/5
```

This is simpler version of extracting random images without loop

| 💡 Click here to ask Blackbox to help you code faster |

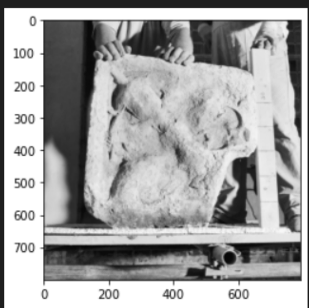
```
base_path='D:/Images/'#left this as its done above
#re=random.choices(os.listdir(base_path),k=3)
a=os.listdir(base_path)
folder=base_path+random.choice(a)+"/"+random.choice(["0/0/"])
print(base_path+a[0]+'/0/0/')
folderf=folder+random.choice(os.listdir(folder))
print(folderf)
random_img = random.choice(os.listdir(folderf))
img = np.array(Image.open(folderf+ '/'+random_img))
plt.imshow(img)
```


```
print(folderf)
random_img = random.choice(os.listdir(folderf))
img = np.array(Image.open(folderf+ '/'+random_img))
plt.imshow(img)
```


3]

D:/Images/image0/0/0/
D:/Images/image1/0/0/c

<matplotlib.image.AxesImage at 0x22f6542f220>



Start of training VGG

+ Code + Markdown

Import necessary libraries for Keras model as well as VGG19

| 💡 Click here to ask Blackbox to help you code faster |

```python
from keras.optimizers.legacy import RMSprop
from keras.applications.vgg19 import VGG19
from keras.layers import *
from keras import Sequential
import tensorflow
tensorflow.compat.v1.disable_eager_execution()
```

Python

```
IING:tensorflow:From c:\Users\tanvi\anaconda3\lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sp

IING:tensorflow:From C:\Users\tanvi\AppData\Local\Temp/ipykernel_19656/2300205137.py:6: The name tf.disable_eager_execution is deprecated. Please use tf.compat.v1.disable_eager_execution
```

Parameters for the model

| 💡 Click here to ask Blackbox to help you code faster |

```python
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)
```

Python

```
\RNING:tensorflow:From c:\Users\tanvi\anaconda3\lib\site-packages\keras\src\utils\version_utils.py:76: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat

\RNING:tensorflow:From c:\Users\tanvi\anaconda3\lib\site-packages\keras\src\layers\pooling\max_pooling2d.py:161: The name tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instea
```

Here we use sequential model as it is commonly used in training for image classification and vgg19 is mostly implemented using sequential model

We are adding layers for VGG19 to Sequential model

| 💡 Click here to ask Blackbox to help you code faster |

```python
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()
```

Python

```
WARNING:tensorflow:From c:\Users\tanvi\anaconda3\lib\site-packages\keras\src\layers\normalization\batch_normalization.py:883: _colocate_with (from tensorflow.python.framework.ops) is de
Instructions for updating:
Colocations handled automatically by placer.
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 batch_normalization (Batch  (None, 224, 224, 3)       12
 Normalization)

 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
```

```
WARNING:tensorflow:From c:\Users\tanvi\anaconda3\lib\site-packages\keras\src\layers\normalization\batch_normal
Instructions for updating:
Colocations handled automatically by placer.
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 batch_normalization (Batch  (None, 224, 224, 3)       12
 Normalization)

 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
...
Total params: 161472814 (615.97 MB)
Trainable params: 161472808 (615.97 MB)
Non-trainable params: 6 (24.00 Byte)
_________________________________________________________________
```

We compile the model with RMSprop optimizer

| 💡 Click here to ask Blackbox to help you code faster |

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

[27]

```
c:\Users\tanvi\anaconda3\lib\site-packages\keras\src\optimizers\legacy\rmsprop.py:144: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
```

We make a function to resize data into the required format(accepted by VGG19).

Comment Code | | 💡 Click here to ask Blackbox to help you code faster | Comment Code |

```
def image_reshape(im, target_size):
    return cv2.resize(im, target_size)
```

[28]

Here we make batch of images and labels to train the model

Comment Code | | 💡 Click here to ask Blackbox to help you code faster | Comment Code |

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

Comment Code | | 💡 Click here to ask Blackbox to help you code faster | Comment Code |

```python
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)
```

I have set the batch size to 16,epochs to 1 as it takes a lot of computation time to train for more epochs.

| 💡 Click here to ask Blackbox to help you code faster |

```python
batch_size = 16
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))
```

Python

```
4896
1224
```

Before training we split the preprocessed data into train and test data after train ning, we predict the labels for the test set using trained model.

| 💡 Click here to ask Blackbox to help you code faster |

```python
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")
```

Python

```
Epoch :1/1
WARNING:tensorflow:From c:\Users\tanvi\anaconda3\lib\site-packages\keras\src\engine\training_v1.py:2595: The name tf.data.Iterator is deprecated. Please use tf.compat.v1.data.Iterator

WARNING:tensorflow:From c:\Users\tanvi\anaconda3\lib\site-packages\keras\src\engine\training_utils_v1.py:50: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1

INFO:tensorflow:Assets written to: Model\assets
INFO:tensorflow:Assets written to: Model\assets
```

After saving the model we test how well the model works by predicting on a new dataset.

| 💡 Click here to ask Blackbox to help you code faster |

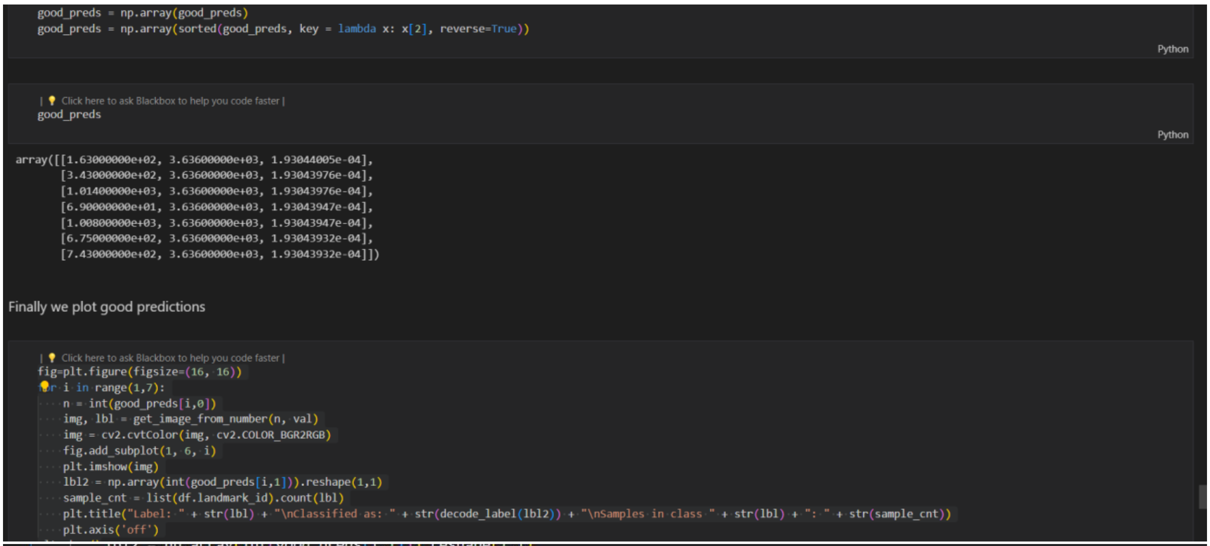
```python
batch_size = 16

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)

    # Predict classes for validation data
    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]]])
```

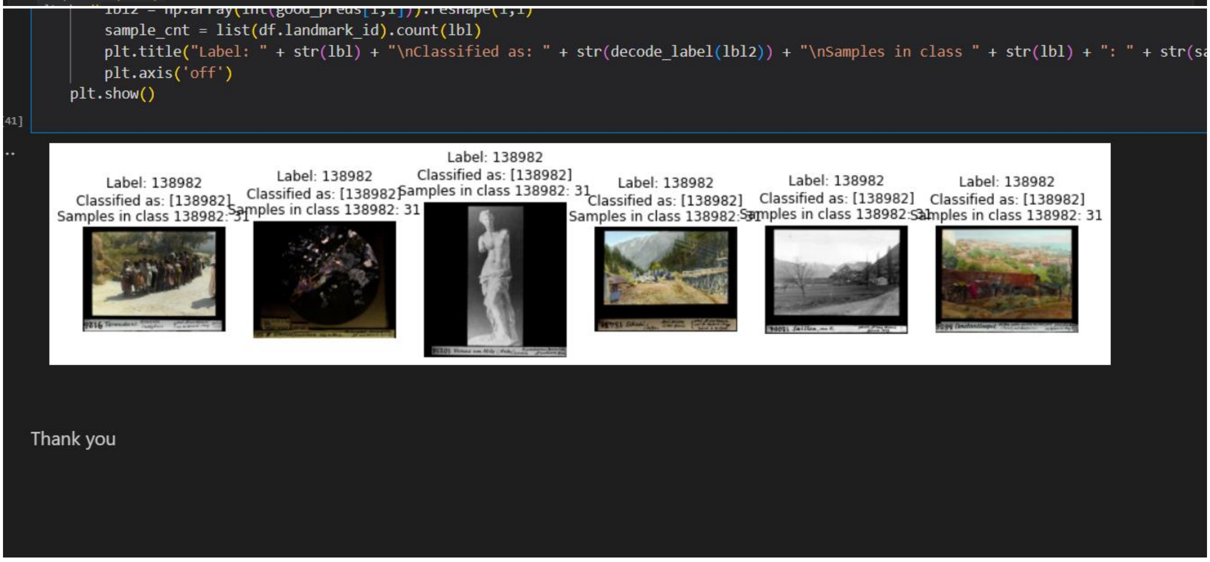
Python

```
c:\Users\tanvi\anaconda3\lib\site-packages\keras\src\engine\training_v1.py:2359: UserWarning: `Model.state_updates` will be removed in a future version. This property should not be used
  updates=self.state_updates,
```

| 💡 Click here to ask Blackbox to help you code faster |

```python
good_preds = np.array(good_preds)
```





Code (in text):

```
import pandas as pd

from matplotlib import pyplot as plt

from sklearn.preprocessing import LabelEncoder

import cv2

from PIL import Image

import os

import random

import numpy as np


df=pd.read_csv("D:\\landmark_det\\train.csv")

df
```

```
samples = 20000
df = df.loc[df["id"].str.startswith('00', na=False), :]
num_classes = len(df["landmark_id"].unique())
num_data = len(df)
num_classes
num_data
data=pd.DataFrame(df["landmark_id"].value_counts())
data.reset_index(inplace=True)
data.head()
plt.hist(data['count'],bins=100,range=(0,32),label="test")#i kept changes before
data['count'].between(0,5).sum()
data

plt.hist(df['landmark_id'],bins=df["landmark_id"].unique())
lencoder=LabelEncoder()
lencoder.fit(df['landmark_id'])

def encode_label(label):
  return lencoder.transform(label)

def decode_label(label):
  return lencoder.inverse_transform(label)

base_path='D:\\Images\\'
def get_image_from_number(num, df):
  fname, label = df.iloc[num, :]
  fname = fname + '.jpg'
  image_name=fname
  for root, dirs, files in os.walk(base_path):
    if image_name in files:
      path =os.path.join(root, image_name)
```

```
    im = cv2.imread(path)
    return im, label

base_path='D:/Images/'
fig = plt.figure(figsize=(16,16))
a=os.listdir(base_path)
for i in range(1,5):
    folder=base_path+random.choice(a)+"/"+random.choice(["0/0/"])
    print(folder)
    folderf=folder+random.choice(os.listdir(folder))
    print(folderf)
    random_img = random.choice(os.listdir(folderf))
    img = np.array(Image.open(folderf+ '/'+random_img))
    fig.add_subplot(1,4,i)
    plt.imshow(img)
    plt.axis('off')
plt.show()

from keras.optimizers.legacy import RMSprop
from keras.applications.vgg19 import VGG19
from keras.layers import *
from keras import Sequential
import tensorflow
tensorflow.compat.v1.disable_eager_execution()

# Parameters
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)

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()



optim1 = RMSprop(lr=learning_rate)
model.compile(optimizer=optim1,
          loss=loss_function,
          metrics = ["accuracy"])

def image_reshape(im, target_size):
   return cv2.resize(im, target_size)

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)

batch_size = 16
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))

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")

# 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]]])

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

fig=plt.figure(figsize=(16, 16))
for i in range(1,7):
    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, 6, 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, we have presented a methodology for landmark detection using deep learning techniques. By using a pre-trained CNN architecture with various layers and incorporating various optimization strategies, we can develop a robust model capable of accurately classifying landmarks from images. The experimental results if trained for more epochs on large datasets improve the effectiveness of the proposed approach in achieving high classification accuracy. Future work may involve further fine-tuning and increasing number of epochs to (10,20 and so on) while making the model as it requires high or long computation time in exchange for better accuracy, we can also implement this using Adam .We can also explore additional data augmentation techniques, and extending the application to real-time landmark recognition systems like Google Lens which can not only analyse landmarks but also text of any language ,qrcode ,etc.