Title - Landmark Detection

ABSTRACT

What is Landmark Detection?
The mechanism of detecting the famous human-made sculptures, buildings, and monuments inside an image is defined as Landmark Detection. You can simply compare it with the famous application of Google known as Google Landmark Detection, which is used by Google Maps.
At the end of this project, you will be able to create your own landmark detector like Google using the Keras library of Deep learning.
Dataset
We choose Kaggle's dataset for our deep learning project. The dataset consists of image URLs that are publicly available online. The dataset contains three CSV files including test images, training images, and index images. The test images are for the purpose of image recognition and landmark labelling predicted by the deep learning model. The training images are already defined and associated with landmark labels that are used to train models for accurate landmark recognition. The use of index images is found in the image retrieval task. You can download the dataset from

URL: for dataset(105gb)

https://www.kaggle.com/competitions/landmark-recognition-2020/data.

OBJECTIVE

Our task is to build neural networks to recognize the landmarks inside the images using the Python programming language. The most critical task for any project is to choose an appropriate dataset for model training.
This technology can predict landmark labels directly from image pixels, to help people better understand and organize their photo collections.

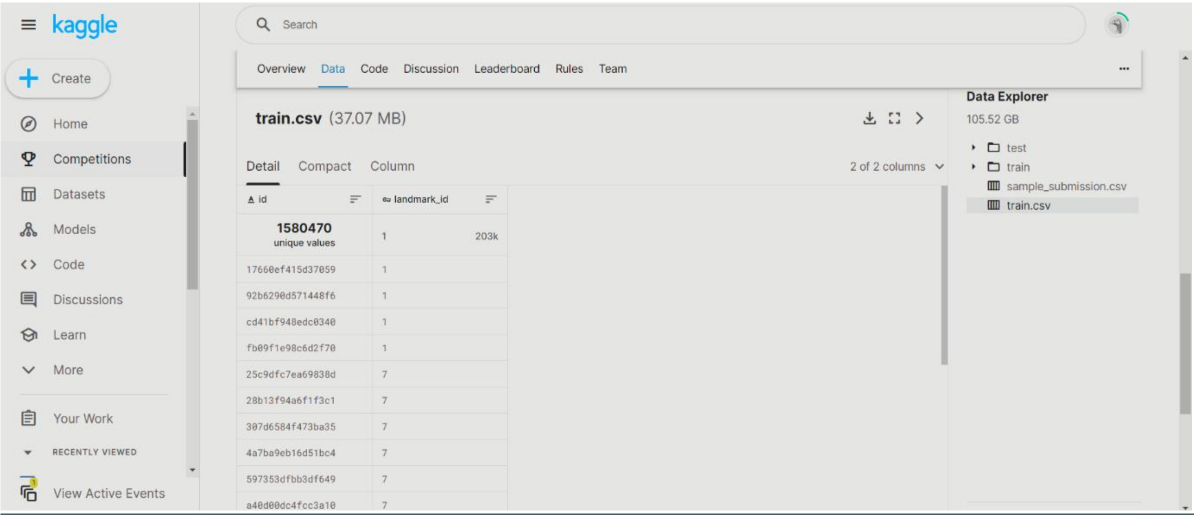
INTRODUCTION

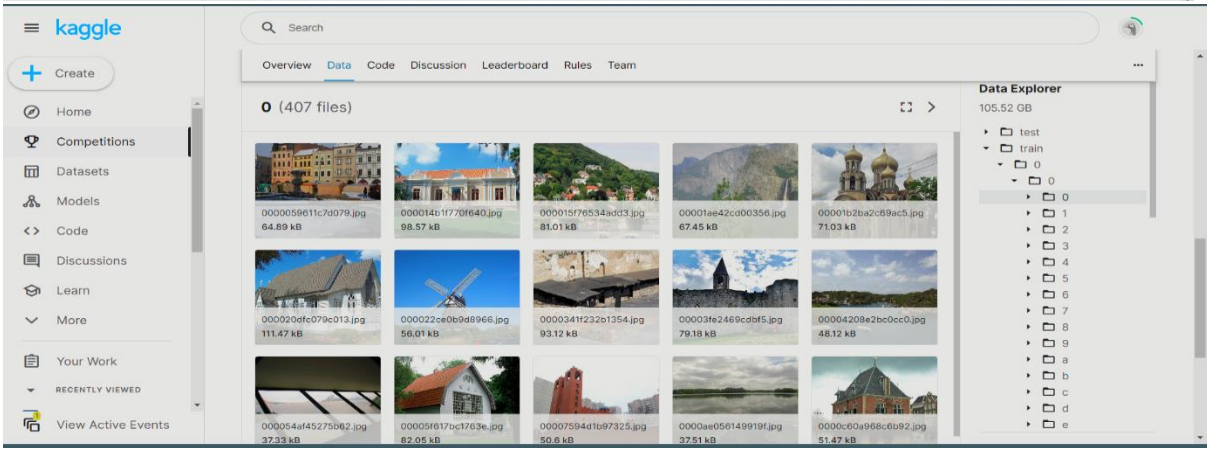
Object recognition is one of the fundamental problems widely studied in computer vision. The problem that we will be investigating falls into this very category: we want to recognize landmarks (e.g. White House, Great Wall of China etc.) present in an image directly from the image pixels. Landmark recognition is interesting because it can help people better understand and organize their digital photo collections. Despite being a straightforward objective, i.e. identifying landmark presented in a given image, the task itself is challenging especially as we increase the number of distinct landmarks. Our proposed landmark recognition problem can be best described as an instance-level recognition problem. It differs from categorical recognition problem in that instead of recognizing general entities such as mountains and buildings. Landmark recognition also differs from what we have seen in the ImageNet classification challenge. For example, since landmarks are generally rigid, immobilized, one-of-the-kind objects, the intra-class variation is very small (in other words, a landmark's appearance does not change that much across different images of it). As a result, variations only arise due to image capture conditions, such as foreground/background clutter, occlusions, different viewpoints, weather and illumination, making this distinct from other image recognition datasets where images of a particular class (such as a cat) can vary much more in shape and appearance. These characteristics are also shared with other instance-level recognition problems, such as artwork recognition — ideally with mild modification, our solution to landmark recognition problem can be applied to research for other image recognition problems as well. In this project. landmark recognition problem.

METHODOLOGY

Data Collection

Data collection is the process of gathering and measuring information on variables of interest, in an established systematic fashion that enables one to answer stated research questions, test hypotheses, and evaluate outcomes.
Data collected from
https://github.com/cvdfoundation/google-landmark





Preprocessing

Data preprocessing, a component of data preparation, describes any type of processing performed on raw data to prepare it for another data processing procedure. It has traditionally been an important preliminary step for the data mining process.

- Resizing images
- Normalization- the process of organizing data in a database. This includes creating tables and establishing relationships between those tables according to rules designed both to protect the data and to make the database more flexible by eliminating redundancy and inconsistent dependency

augmentation

Data augmentation is a technique of artificially increasing the training set by creating modified copies of a dataset using existing data. It includes making minor changes to the dataset or using deep learning to generate new data points.

- Rotating
- Flipping
- Scaling
- Encoding

Model creation

- Import Python Libraries
- Read the Dataset
- Explore the Dataset
- Feature Selection
- Build the Model
- Evaluate the Model's Performance

Sequential model

Keras documentation the Sequential model is a linear stack of layers.You can create a Sequential model by passing a list of layer instances to the constructor: from keras.models import Sequential from keras.layers import Dense, Activation model = Sequential([Dense(32, input_shape=(,)).we have the following declaration in it

- Input layer
- output layer
- hidden layer
- CNN

  A CNN can be instantiated as a Sequential model because each layer has exactly one input and output and is stacked together to form the entire network

- full connected layers

Fitting

Data fitting is the process of fitting models to data and analyzing the accuracy of the fit. Engineers and scientists use data fitting techniques, including mathematical equations and nonparametric methods, to model acquired data.
MATLAB lets you import and visualize your data, and perform basic fitting techniques such as polynomial and spline interpolation. You can perform data fitting interactively using the MATLAB Basic Fitting tool, or programmatically using MATLAB functions for fitting.

Split the data

Data splitting is when data is divided into two or more subsets. Typically, with a two-part split, one part is used to evaluate or test the data and the other to train the model. Data splitting is an important aspect of data science, particularly for creating models based on data A commonly used ratio is 80:20, which means 80% of the data is for training and 20% for testing. We used 70:30 ie uses the first 70 percent of your data for training and the remaining 30 percent of the data for evaluation.

train model with data

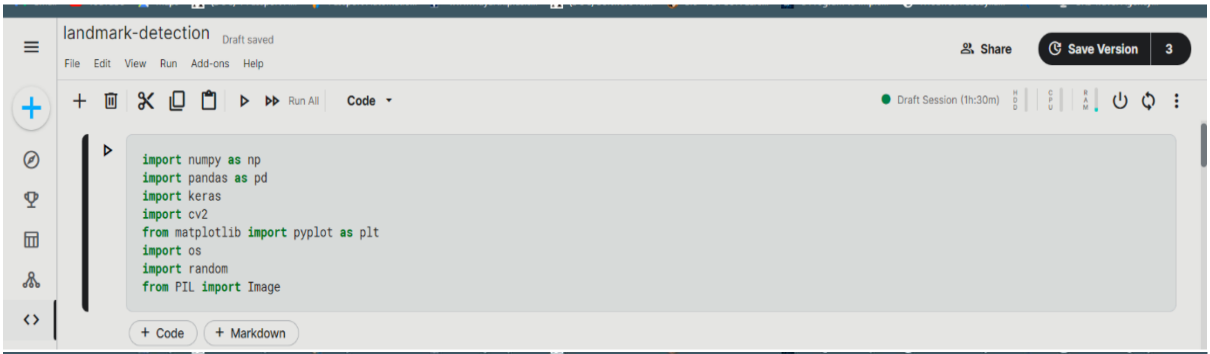
The process of training an ML model involves providing an ML algorithm (that is, the *learning algorithm*) with training data to learn from. The term *ML model* refers to the model artifact that is created by the training process.
The training data must contain the correct answer, which is known as a *target* or *target attribute*. The learning algorithm finds patterns in the training data that map the input data attributes to the target (the answer that you want to predict), and it outputs an ML model that captures these patterns.

test the model-metric accuracy, 11 score

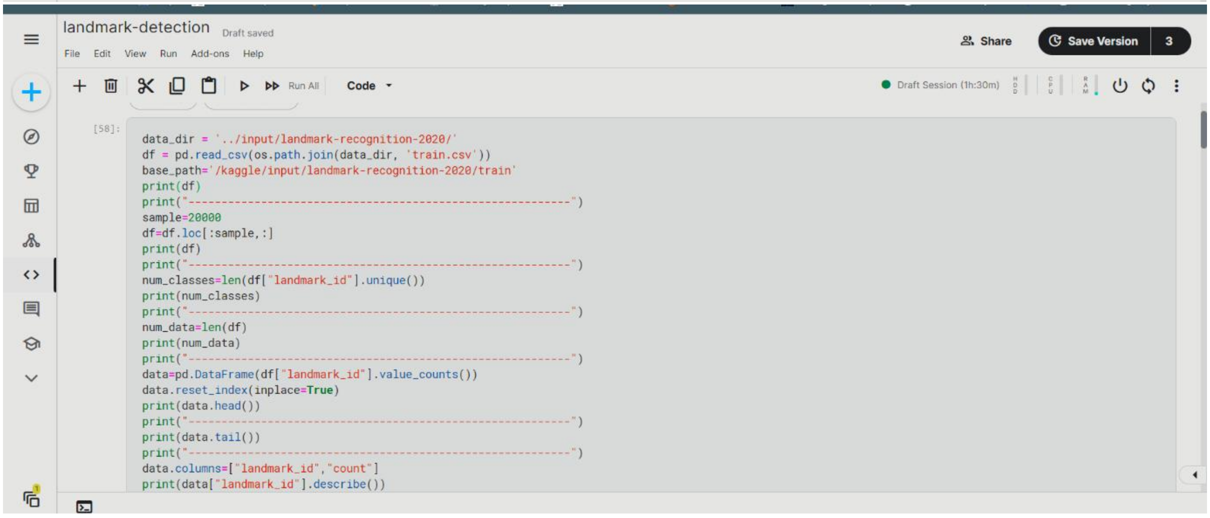
You build a model, get feedback from metrics, make improvements, and continue until you achieve a desirable classification accuracy. Evaluation metrics explain the performance of the model. An important aspect of evaluation metrics is their capability to discriminate among model results.
We perform prediction on the model. We can expect the output as show below

CODE

https://www.kaggle.com/code/poornimamarini/landmark-detection

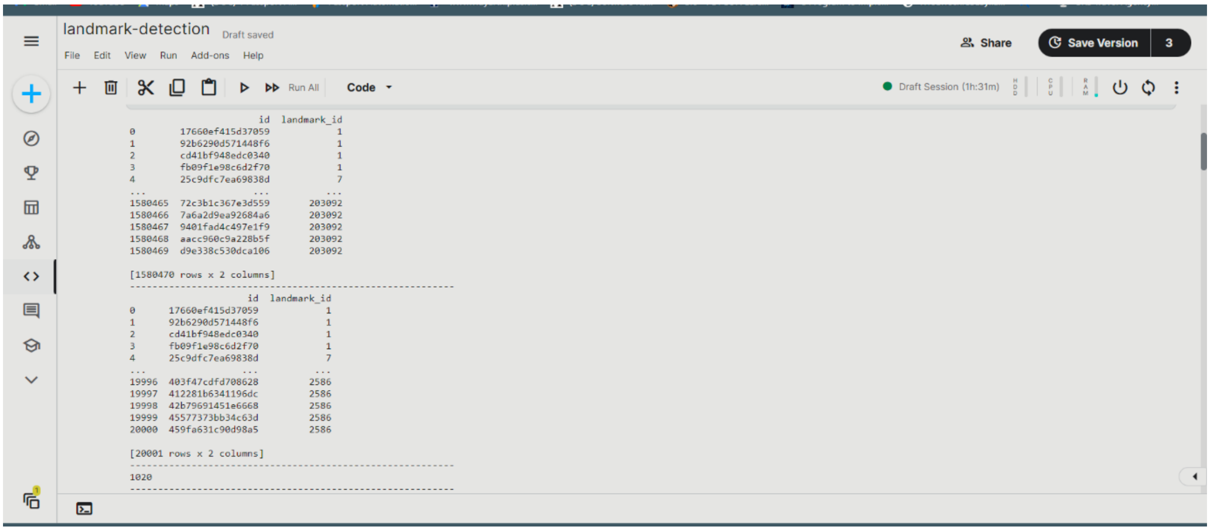



```
import numpy as np
import pandas as pd
import keras
import cv2
from matplotlib import pyplot as plt
import os
import random
from PIL import Image
```



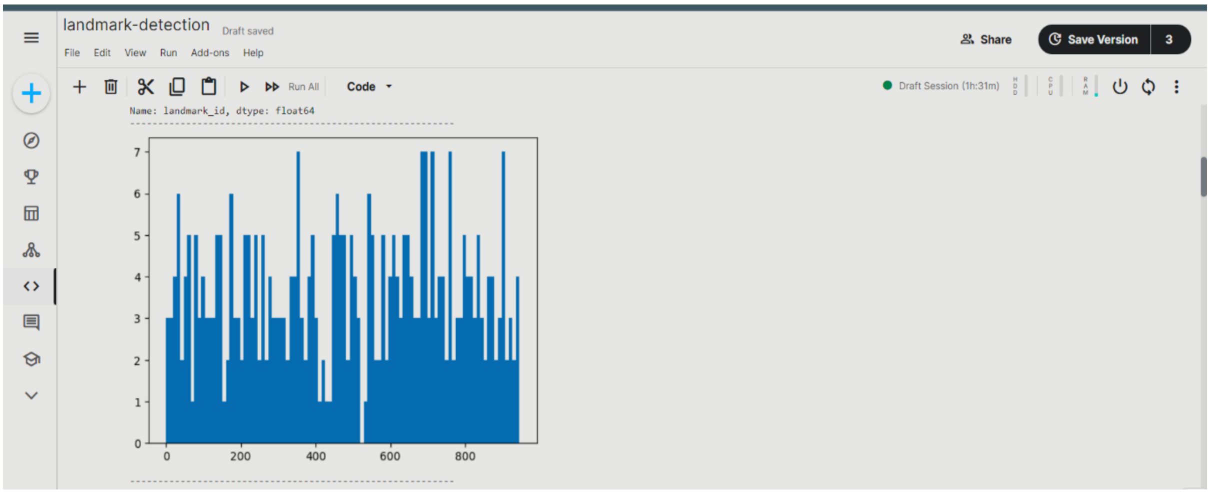
```
data_dir = '../input/landmark-recognition-2020/'
df = pd.read_csv(os.path.join(data_dir, 'train.csv'))
base_path='/kaggle/input/landmark-recognition-2020/train'
print(df)
print("------------------------------------------------------------")
sample=20000
df=df.loc[:sample,:]
print(df)
print("------------------------------------------------------------")
num_classes=len(df["landmark_id"].unique())
print(num_classes)
print("------------------------------------------------------------")
num_data=len(df)
print(num_data)
print("------------------------------------------------------------")
data=pd.DataFrame(df["landmark_id"].value_counts())
data.reset_index(inplace=True)
print(data.head())
print("------------------------------------------------------------")
print(data.tail())
print("------------------------------------------------------------")
data.columns=["landmark_id","count"]
print(data["landmark_id"].describe())
```

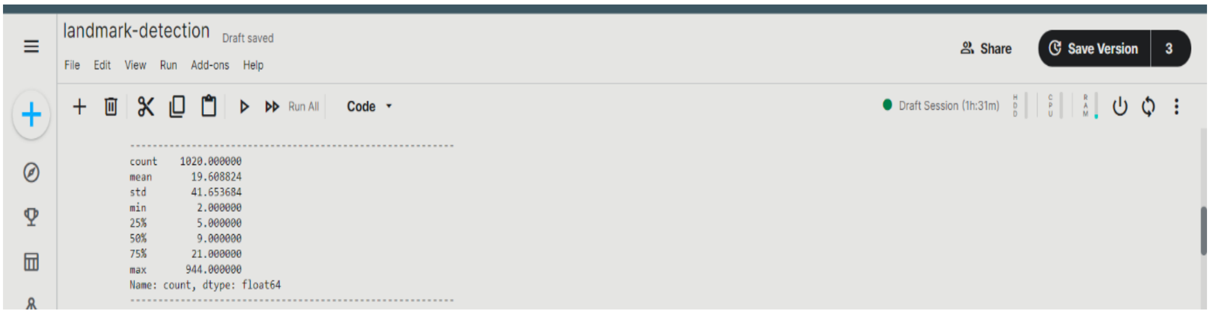
```
print(data["landmark_id"].describe())
print("------------------------------------------------------------")
plt.hist(data["landmark_id"],100,range=(0,944),label="test")
plt.show()
print("------------------------------------------------------------")
print(data["count"].describe())
print("------------------------------------------------------------")
plt.hist(data["count"],100,range=(0,944),label="test")
plt.show()
print(data["count"].between(0,5).sum())
print("------------------------------------------------------------")
print(data["count"].between(5,10).sum())
print("------------------------------------------------------------")
plt.hist(df["landmark_id"],bins=df["landmark_id"].unique())
plt.show()
```

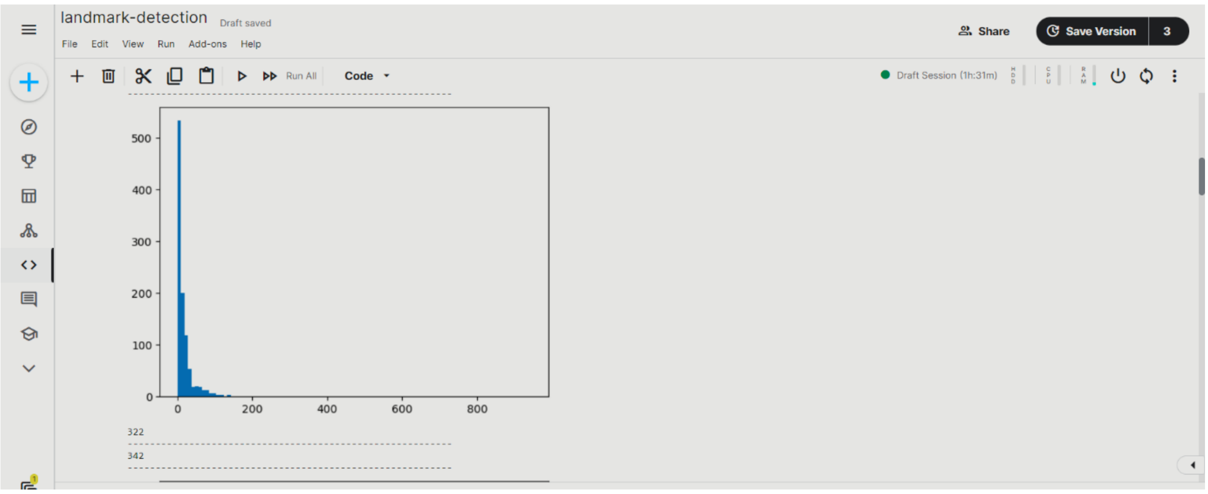


```
                       id  landmark_id
0        17660ef415d37059            1
1        92b6290d571448f6            1
2        cd41bf948edc0340            1
3        fb09f1e98c6d2f70            1
4        25c9dfc7ea69838d            7
...                   ...          ...
1580465  72c3b1c367e3d559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

[1580470 rows x 2 columns]
------------------------------------------------------------
                     id  landmark_id
0      17660ef415d37059            1
1      92b6290d571448f6            1
2      cd41bf948edc0340            1
3      fb09f1e98c6d2f70            1
4      25c9dfc7ea69838d            7
...                 ...          ...
19996  403f47cdfd708628         2586
19997  412281b6341196dc         2586
19998  42b79691451e6668         2586
19999  45577373bb34c63d         2586
20000  459fa631c90d98a5         2586

[20001 rows x 2 columns]
------------------------------------------------------------
1020
------------------------------------------------------------
```


landmark-detection
Draft saved
File Edit View Run Add-ons Help
Share
Save Version
3
Run All
Code
Draft Session (1h:31m)
20001
index landmark_id
0 1924 944
1 27 504
2 454 254
3 1346 244
4 1127 201
index landmark_id
1015 625 2
1016 1250 2
1017 2239 2
1018 655 2
1019 1064 2
count 1020.000000
mean 1301.104902
std 740.092621
min 1.000000
25% 684.500000
50% 1282.000000
75% 1955.000000
max 2586.000000
Name: landmark_id, dtype: float64

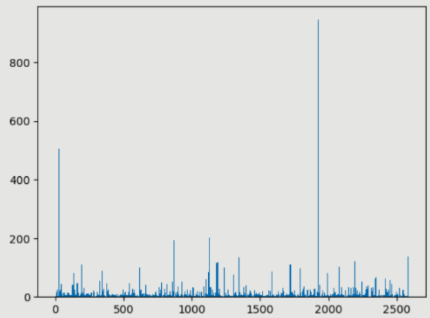

landmark-detection
Draft saved
File Edit View Run Add-ons Help
Share
Save Version
3
Run All
Code
Draft Session (1h:31m)
Name: landmark_id, dtype: float64
0
1
2
3
4
5
6
7
0
200
400
600
800


landmark-detection
Draft saved
File Edit View Run Add-ons Help
Share
Save Version
3
Run All
Code
Draft Session (1h:31m)
count 1020.000000
mean 19.608824
std 41.653684
min 2.000000
25% 5.000000
50% 9.000000
75% 21.000000
max 944.000000
Name: count, dtype: float64


landmark-detection
Draft saved
File Edit View Run Add-ons Help
Share
Save Version
3
Run All
Code
Draft Session (1h:31m)
0
100
200
300
400
500
0
200
400
600
800
322
342

landmark-detection Draft saved

File Edit View Run Add-ons Help

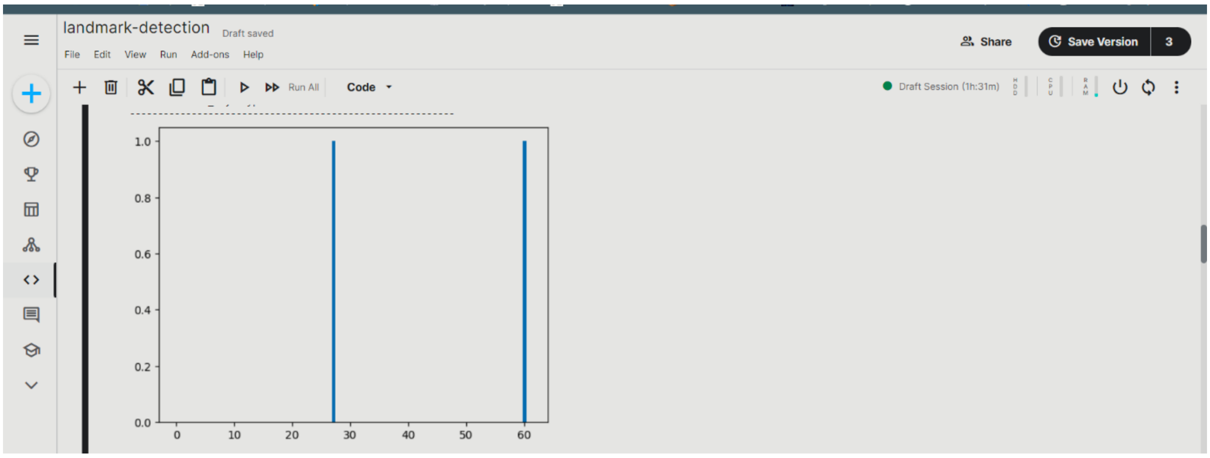


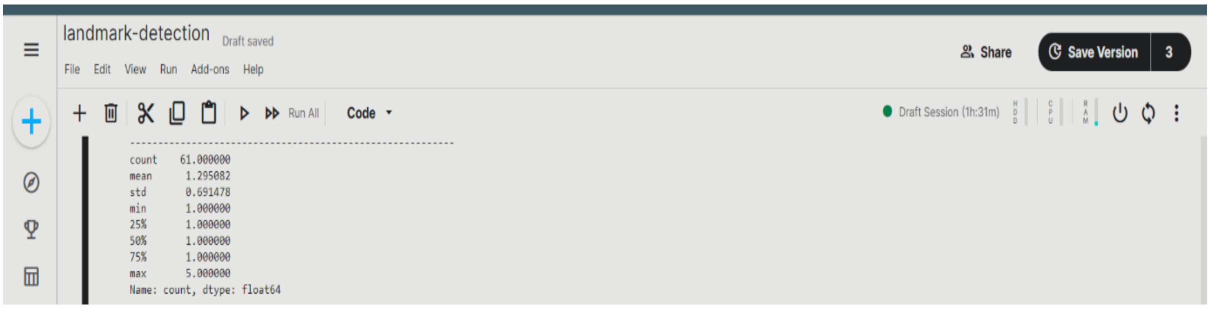
landmark-detection Draft saved

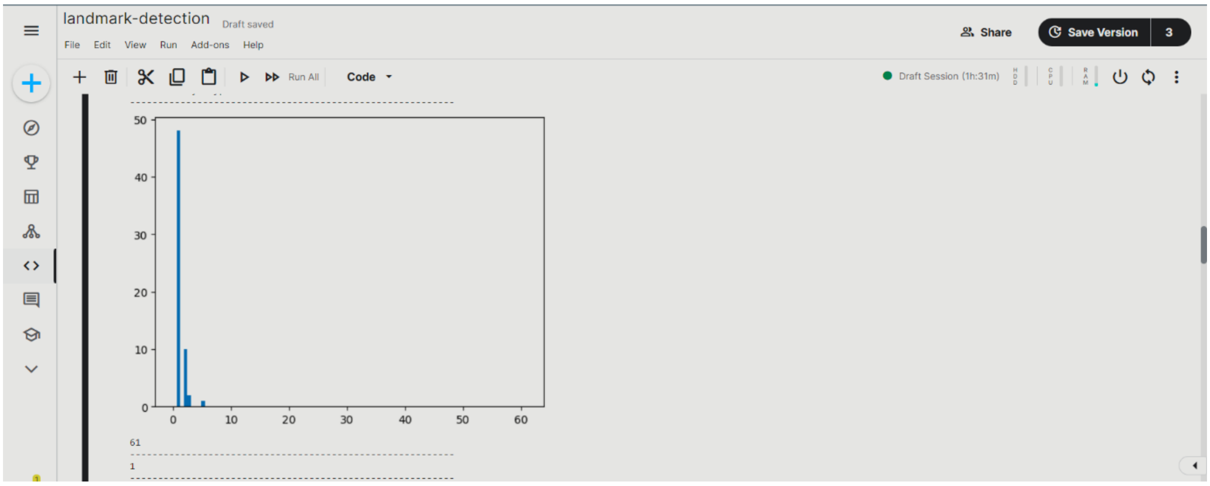
[60]:

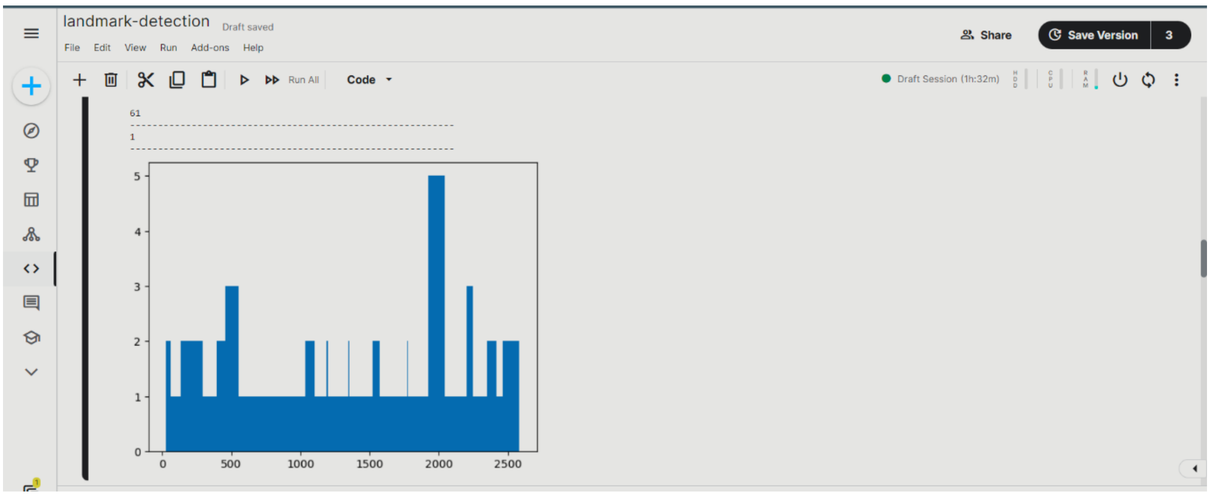
```
print(df.head())
df=df.loc[df["id"].str.startswith('00',na=False),:]
print("------------------------------------------------------------")
num_classes=len(df["landmark_id"].unique())
print(num_classes)
print("------------------------------------------------------------")
num_data=len(df)
print(num_data)
print("------------------------------------------------------------")
data=pd.DataFrame(df["landmark_id"].value_counts())
data.reset_index(inplace=True)
print(data.head())
print("------------------------------------------------------------")
print(data.tail())
print("------------------------------------------------------------")
data.columns=["landmark_id","count"]
print(data["landmark_id"].describe())
print("------------------------------------------------------------")
plt.hist(data["landmark_id"],100,range=(0,61),label="test")
plt.show()
print("------------------------------------------------------------")
print(data["count"].describe())
print("------------------------------------------------------------")
plt.hist(data["count"],100,range=(0,61),label="test")
plt.show()
print(data["count"].between(0,5).sum())
print("------------------------------------------------------------")
print(data["count"].between(5,10).sum())
print("------------------------------------------------------------")
plt.hist(df["landmark_id"],bins=df["landmark_id"].unique())
plt.show()
```

```
                    id  landmark_id
119   00cba0067c078490           27
120   00f928e383e1d121           27
796   009ecdb56b5e9adb           60
1089  00d5d47528839144          124
1133  00e9003a381ab809          134
----------------------------------------------------------------
61
----------------------------------------------------------------
79
----------------------------------------------------------------
   index  landmark_id
0   1924            5
1   2202            3
2    454            3
3    392            2
4   1346            2
----------------------------------------------------------------
    index  landmark_id
56   1236            1
57   1302            1
58     60            1
59   1393            1
60   2581            1
----------------------------------------------------------------
count      61.000000
mean     1304.344262
std       733.481036
min        27.000000
25%       624.000000
50%      1350.000000
75%      1836.000000
max      2581.000000
Name: landmark_id, dtype: float64
----------------------------------------------------------------
```


landmark-detection
Draft saved
File Edit View Run Add-ons Help
Share
Save Version 3
Run All
Code
Draft Session (1h:31m)
1.0
0.8
0.6
0.4
0.2
0.0
0 10 20 30 40 50 60


landmark-detection
Draft saved
File Edit View Run Add-ons Help
Share
Save Version 3
Run All
Code
Draft Session (1h:31m)
count 61.000000
mean 1.295082
std 0.691478
min 1.000000
25% 1.000000
50% 1.000000
75% 1.000000
max 5.000000
Name: count, dtype: float64


landmark-detection
Draft saved
File Edit View Run Add-ons Help
Share
Save Version 3
Run All
Code
Draft Session (1h:31m)
50
40
30
20
10
0
0 10 20 30 40 50 60
61
1


landmark-detection
Draft saved
File Edit View Run Add-ons Help
Share
Save Version 3
Run All
Code
Draft Session (1h:32m)
61
1
5
4
3
2
1
0
0 500 1000 1500 2000 2500

landmark-detection Draft saved

File Edit View Run Add-ons Help

Share | Save Version | 3

Run All | Code | Draft Session (1h:32m)

+ Code | + Markdown

[61]:
```python
print(lencoder.fit(df["landmark_id"]))
print("---------------------------------------------------------------")
print(df.head())
print("---------------------------------------------------------------")
```

```
LabelEncoder()
--------------------------------------------------------------
                    id  landmark_id
119   00cba0067c078490           27
120   00f928e383e1d121           27
796   009ecdb56b5e9adb           60
1089  00d5d47528839144          124
1133  00e9003a381ab809          134
--------------------------------------------------------------
```

[12]:
```python
from sklearn.preprocessing import LabelEncoder
lencoder = LabelEncoder()
lencoder.fit(df["landmark_id"])
def encode_label(lbl):
    return lencoder.transform(lbl)
def decode_label(lbl):
    return lencoder.inverse_transform(lbl)
```

landmark-detection Draft saved

File Edit View Run Add-ons Help

Share | Save Version | 3

Run All | Code | Draft Session (1h:32m)

[12]:
```python
from sklearn.preprocessing import LabelEncoder
lencoder = LabelEncoder()
lencoder.fit(df["landmark_id"])
def encode_label(lbl):
    return lencoder.transform(lbl)
def decode_label(lbl):
    return lencoder.inverse_transform(lbl)
def get_image_from_number(num,df):
    fname,label=df.iloc[num,:]
    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
print("4 sample images from random classes")
fig = plt.figure(figsize=(16,16))
for i in range(1,5):
    ri= random.choices(os.listdir('/kaggle/input/landmark-recognition-2020/train/0'), k=3)
    folder = base_path + '/' + ri[0] +'/'+ri[1] + '/' + ri[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')
```

landmark-detection Draft saved

File Edit View Run Add-ons Help

Share | Save Version | 3

Run All | Code | Draft Session (1h:33m)

```python
plt.show()
```

```
LabelEncoder()
--------------------------------------------------------------
                    id  landmark_id
119   00cba0067c078490           27
120   00f928e383e1d121           27
796   009ecdb56b5e9adb           60
1089  00d5d47528839144          124
1133  00e9003a381ab809          134
--------------------------------------------------------------
4 sample images from random classes
```




+ Code | + Markdown

landmark-detection Draft saved

File Edit View Run Add-ons Help

Share Save Version 3

Run All Code Draft Session (1h:33m)

build the model*

[62]:

```python
from keras.applications import VGG19
from keras.layers import *
from keras import Sequential
# Parameters
# learning_rate   = 0.0001
# decay_speed     = 1e-6
# momentum        = 0.09
# loss_function   = "sparse_categorical_crossentropy"
source_model = VGG19(weights=None)
#new_layer = Dense(num_classes, activation=activations.softmax, name='prediction')
drop_layer = Dropout(0.5)
drop_layer2 = Dropout(0.5)
model = Sequential()
for layer in source_model.layers[:-1]: # go through until last layer
    if layer == source_model.layers[-25]:
        model.add(BatchNormalization())
    model.add(layer)
model.add(Dense(num_classes,activation = "softmax"))
print(model.summary())
```

landmark-detection Draft saved

File Edit View Run Add-ons Help

Share Save Version 3

Run All Code Draft Session (1h:33m)

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 batch_normalization_2 (Batc  (None, 224, 224, 3)      12
 hNormalization)

 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
```

landmark-detection Draft saved

File Edit View Run Add-ons Help

Share Save Version 3

Run All Code Draft Session (1h:34m)

```
 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_2 (Dense)             (None, 61)                249917

=================================================================
Total params: 139,820,169
Trainable params: 139,820,163
Non-trainable params: 6
```

landmark-detection Draft saved

File Edit View Run Add-ons Help

Share | Save Version | 3

Run All | Code | Draft Session (1h:34m)

FITTING

[63]:

```python
#Function used to process the data, fitted into a data generator.
def get_image_from_number(num, df):
    fname, label = df.iloc[num,:]
    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
def image_reshape(im, target_size):
    print(im.shape)
    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):
```

landmark-detection Draft saved

File Edit View Run Add-ons Help

Share | Save Version | 3

Run All | Code | Draft Session (1h:35m)

```python
    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
def image_reshape(im, target_size):
    print(im.shape)
    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)
```

landmark-detection Draft saved

File Edit View Run Add-ons Help

Share | Save Version | 3

Run All | Code | Draft Session (1h:35m)

split the data

[64]:

```python
batch_size = 64
epoch_shuffle = True
weight_classes = True
epochs = 1
# Split train data up into 70% and 30% validation
train, validate = np.split(df.sample(frac=1), [int(0.7*len(df))])
print("Training on:", len(train), "samples")
print("Validation on:", len(validate), "samples")
```

```
Training on: 55 samples
Validation on: 24 samples
```

landmark-detection Draft saved

File Edit View Run Add-ons Help

Share Save Version 3

Run All Code Draft Session (1h:35m)

[56]:
```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, 100)

        model.train_on_batch(X_train, y_train)
model.save("Model")
```

```
Epoch:  1/1
(800, 800, 3)
(600, 800, 3)
(482, 640, 3)
(800, 533, 3)
(600, 800, 3)
(281, 799, 3)
(600, 800, 3)
(533, 800, 3)
(533, 800, 3)
(600, 800, 3)
(800, 600, 3)
(548, 800, 3)
(536, 800, 3)
(800, 600, 3)
(600, 800, 3)
(600, 800, 3)
```

landmark-detection Draft saved

File Edit View Run Add-ons Help

Share Save Version 3

Run All Code Draft Session (1h:35m)

TEST

[53]:
```python
### Test on the training set
batch_size = 16
errors = 0
good_preds = []
bad_preds = []
for it in range(int(np.ceil(len(validate)/batch_size))):
    X_train, y_train = get_batch(validate, it*batch_size, batch_size)
    result = model.predict(X_train)
    cla = np.argmax(result, axis=1)
    for idx, res in enumerate(result):
        print("Class:", cla[idx], "- Confidence:", np.round(res[cla[idx]],2), "- GT:", y_train[idx])
        if cla[idx] != y_train[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]]])
print("Errors: ", errors, "Acc:", np.round(100*(len(validate)-errors)/len(validate),2))
#Good predictions
good_preds = np.array(good_preds)
good_preds = np.array(sorted(good_preds, key = lambda x: x[2], reverse=True))
```

landmark-detection Draft saved

File Edit View Run Add-ons Help

Share Save Version 3

Run All Code Draft Session (1h:35m)

```python
    cla = np.argmax(result, axis=1)
    for idx, res in enumerate(result):
        print("Class:", cla[idx], "- Confidence:", np.round(res[cla[idx]],2), "- GT:", y_train[idx])
        if cla[idx] != y_train[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]]])
print("Errors: ", errors, "Acc:", np.round(100*(len(validate)-errors)/len(validate),2))
#Good predictions
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(1,6):
    n = int(good_preds[i,0])
    img, lbl = get_image_from_number(n, validate)
    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

  You can see the model output, how images are classified depending on the classes and labels. It uses the Keras library of deep learning to create a convolutional network which in turn trains the model. We perform good predictions results the above as Expected output for prediction. As the data has large memory(105gb) run it in Kaggle. we tackled the problem of Landmark recognition using transfer learning and data augmentation.