CapstoneProject

April 27, 2018

1 Project: Google Landmark Recognition Challenge

1.1 Domain Background

The Google landmark Recognition Challenge on Kaggle is selected to be a capstone project [1]. This topic is interested to me not only that the image recognition is a fundamental problem in computer vision of machine learning but also that this type of question is always faced in the real world. It is a good topic to a machine-learning beginner because we have learned similar problem in Udacity classes. However, how to apply what we learned into a real case is a totally different scenario. If we can try to solve this type of problem, it will give us an experience of interacting and dealing with a real world case during the problem solving process. On the other hand, landmark recognition is useful to remind users what it is or introduce it to travelers who pass by. For example, travelers took a picture with a building, but forgot the name of this building when they try to put some information on social media or share to their friends. By using this recognition tool, it is easy to remind travelers what the building is. Moreover, when people see a landmark, they would like to know more information of it. This machine-learning model can recognize this image and provides a specifiec information to people. ## Problem Statement The landmark recognition is a typical problem of computer vision. However, it is a little different from the regular problem. In this case, there are 15k classes of landmarks. The traditional image recognition on ImageNet is able to recognize 1k classes. That means the difficulty is how to recognize this large number of classes. The basic idea is building a CNN model and trains this model, but it causes much computation time to reach an acceptable accuracy. The potential solution is using a better architecture or transferring learning to reduce the computation time and get higher accuracy. ## Evaluation Metrics The evaluation metrics is Global Average Precession (GAP) at k, where k=1 [3]. For each image, the model has to predict its landmark with confidence score. If there are N predictions for one image, the predictions should be sorted by their confidence score and apply to a function as follows. The evaluation function will not be implemented becasue the test result will send to Kaggle and Kaggle will provide the test GAP.

$$GAP = \frac{1}{M}\sum_{i=1}^{N} P(i)rel(i)$$

where: * N is the total number of predictions returned by the system, across all queries * M is the total number of queries with at least one landmark from the training set visible in it (note that some queries may not depict landmarks) * P(i) is the precision at rank i * Rel(i) denotes the relevance of prediction i: it's 1 if the i-th prediction is correct and 0 otherwise

2 Analysis

2.1 Data Exploration

Before training a model, a data exploration has to be applied. First, using basic pandas function obtains datasets' information such as data size, format, and data distribution.

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import os
        import math
        from tqdm import tqdm
        from keras.utils import np_utils
        %matplotlib inline
```

```
C:\Users\Gamer\Anaconda3\envs\tfgpu\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conve
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

Load dataset and check the size of train data and test data whichi is provided by the website.

```
In [2]: # load train csv file
        train_csv = pd.read_csv('data/train/train.csv')
        # show the size of the train data
        print(train_csv.shape)
        # load test csv file
        test_csv = pd.read_csv('data/test/test.csv')
        # show the size of the test data
        print(test_csv.shape)
```

```
(1225029, 3)
(117703, 2)
```

Display the data set and show some data samples.

```
In [3]: train_csv.head(5)
```

```
Out[3]:                  id                                                url  \
        0  cacf8152e2d2ae60  http://static.panoramio.com/photos/original/70...
        1  0a58358a2afd3e4e  http://lh6.ggpht.com/-igpT6wu0mIA/ROV8HnUuABI/...
        2  6b2bb500b6a38aa0  http://lh6.ggpht.com/-vKr5G5MEusk/SR6r6SJi6mI/...
        3  b399f09dee9c3c67  https://lh3.googleusercontent.com/-LOW2cjAqubA...
        4  19ace29d77a5be66  https://lh5.googleusercontent.com/-tnmSXwQcWL8...

           landmark_id
        0         4676
        1         6651
        2        11284
        3         8429
        4         6231
```

```
In [4]: test_csv.head()
```

```
Out[4]:                 id                                                url
        0  000088da12d664db  https://lh3.googleusercontent.com/-k45wfamuhT8...
        1  0001623c6d808702  https://lh3.googleusercontent.com/-0Q0ywv8KVIA...
        2  0001bbb682d45002  https://lh3.googleusercontent.com/-kloLenz1xZk...
        3  0002362830cfe3a3  https://lh3.googleusercontent.com/-N6z79jNZYTg...
        4  000270c9100de789  https://lh3.googleusercontent.com/-keriHaV0q1U...
```

According to the above information, the train dataset contains three attributes, id, url, and landmark_id. The following is an example of train dataset.

id	url	landmark_id
cacf8152e2d2ae60	http://static.panoramio.com/photos/original/70...	4676

The test dataset contains two attibutes, id and url. The test dataset is shown as follows:

id	url
000088da12d664db	https://lh3.googleusercontent.com/-k45wfamuhT8...

That means images are downloaded through these urls and tag these images with landmark_id for train datasets. The real train and test images should be downloaded separately.

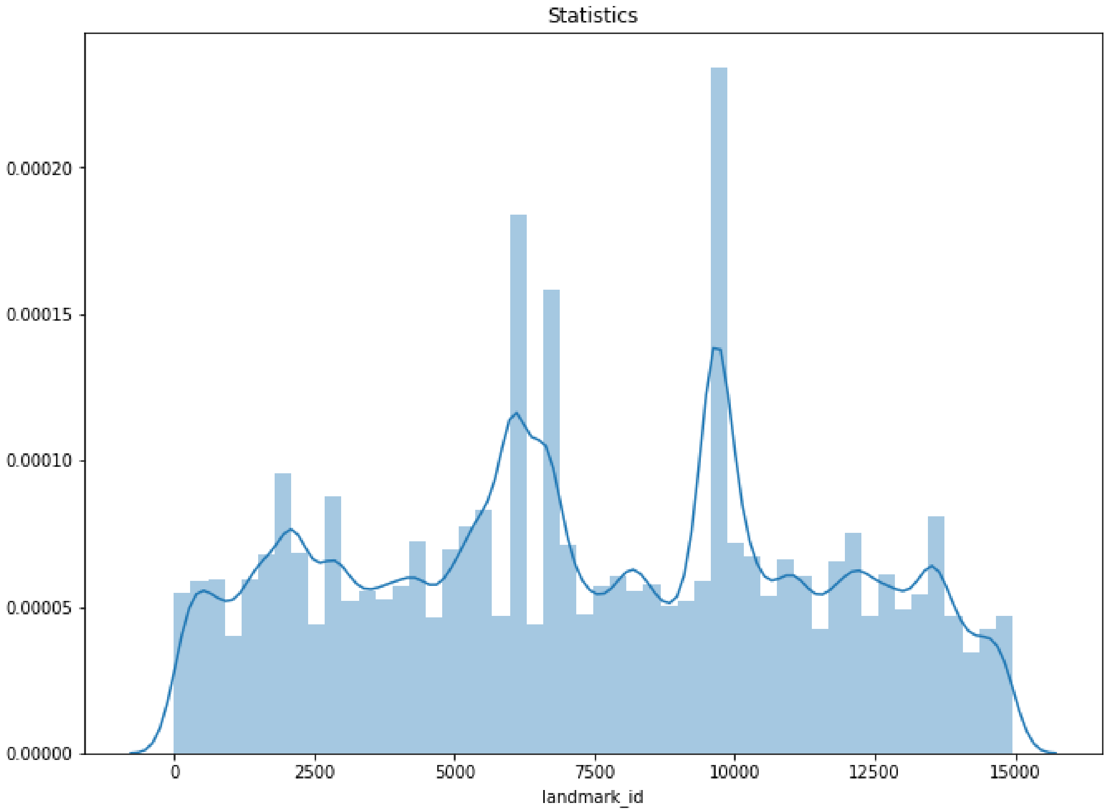
Check the landmark train data statistics to see if there is abnormal datasets.

```
In [5]: # get the image frequent of each landmark
        train_stat = pd.DataFrame(train_csv.landmark_id.value_counts())
        train_stat.reset_index(inplace=True)
        train_stat.columns=['landmark_id','count']
        print(train_stat.shape)
        print('### Most Frequent landmark_id ###')
        print(train_stat.head())
        print('### Least Frequent landmark_id ###')
        print(train_stat.tail())
        # plot image frequent
        plt.figure(figsize = (11, 8))
        plt.title('Statistics')
        sns.distplot(train_csv['landmark_id'])
        plt.show()
```

```
(14951, 2)
### Most Frequent landmark_id ###
   landmark_id  count
```

```
0          9633  50337
1          6051  50148
2          6599  23415
3          9779  18471
4          2061  13271
### Least Frequent landmark_id ###
       landmark_id  count
14946         1527      1
14947         6025      1
14948         4334      1
14949         5865      1
14950         8381      1
```

```
C:\Users\Gamer\Anaconda3\envs\tfgpu\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarnin
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



According to the above frequence chart of landmarks, it shows that some landmark contain many images for training bu t some landmarks contain only one image. This train dataset is highly skewed.

2.2 Exploratory Visualization

Defore downloading all datasets, some images will be downloaded first to show image specification.

```
In [6]: from IPython.display import Image
        from IPython.core.display import HTML

        def category_image(urls):
            imagesHTML = ''.join(["<img style='{img_style}' src='{u}' />".
                                  format(img_style="width: 180px; margin: 0px; float: left; bo
                                         u=val)
                                  for key, val in urls.iteritems()])
            display(HTML(imagesHTML))

        urls = train_csv['url'][100:105]
        category_image(urls)

<IPython.core.display.HTML object>
```

After exploration the dataset, images contains various dimensions so a resizing function has to be applied before training the model. There are 1,225,029 images and 14,951 classes in the training dataset. However, the image examples are not balance. Some landmarks only contain one image, and some landmarks contain more than 50k images.

2.2.1 Download train and test data

There is a saperate code in this folder called d.py.
This program will donwnload the images from website and resize image dimensions to 224 X 224.
Also, it creates a trainImageList for train images which downloaded successfully, and a testImageList file for test images which downloaded successfully.
The dataset is gigantic, so there are only 50% downloaded.

```
In [3]: train = pd.read_csv('data/trainImagesList')
        display(train.head(5))
        display(train.shape)

                 id  landmark_id                              filename
0  4cddf5dfec480378        10900  data/trainImages/4cddf5dfec480378.jpg
1  e892105697730cd0         9633  data/trainImages/e892105697730cd0.jpg
2  e6ca7e6d1fb0c30e         7979  data/trainImages/e6ca7e6d1fb0c30e.jpg
3  5b7e170e3f82df79         8487  data/trainImages/5b7e170e3f82df79.jpg
4  8bb30ed8ded320b5        10045  data/trainImages/8bb30ed8ded320b5.jpg


(593365, 3)
```

```
In [2]: test = pd.read_csv('data/testImagesList')
        display(test.head(5))
        display(test.shape)
```

```
                 id landmark_id                                filename
0  000088da12d664db        None  data/testImages/000088da12d664db.jpg
1  0001623c6d808702        None  data/testImages/0001623c6d808702.jpg
2  0001bbb682d45002        None  data/testImages/0001bbb682d45002.jpg
3  0002362830cfe3a3        None  data/testImages/0002362830cfe3a3.jpg
4  000270c9100de789        None  data/testImages/000270c9100de789.jpg


(115698, 3)
```

When downloading, there is a image list file containing imgae infornation whichh has downlaoded successfully. In the image list file, there id, landmark_id and filename. The id and landmark_id are the same with the original dataset's. The onnly difference is that url is replaced by a filename. The filename can be used to get the downloaded image.

id	landmark_id	filename
cacf8152e2d2ae60	10900	data/trainImages/4cddf5dfec480378.jpg

For the test dataset, there is no landmark_id.

id	landmark_id	filename
0002362830cfe3a3	None	data/testImages/0002362830cfe3a3.jpg

2.3 Algorithms and Techniques

Algorithms and techniques used in the project are thoroughly discussed and properly justified based on the characteristics of the problem.
This is a typical image recgonition problem, so a CNN method will be applied in this project. The following steps will be implemented.
* Before training the data, there are several steps to preprocess data. * Select 100 examples from each class. If a class contains more than 100 examples, randomly select 100 of them. If a class contains less than 100 examples, use image generator (rotate, shift and zoom) creating examples to meet 100 training examples. * Image size will be normalized to 224 X 224. * Using the benchmark model gets a GAP score as a benchmark score. * Implement a basic CNN architecture * One convolution layer and one maxpooling layer repeat three times and the model ends with a fully connected layer. * Implement transferring leaning. There four architecture will be tried, VGG-19, ResNet-50, Inception, and Xception. * Compare the results of two architectures. * Improve the GAP by using a better architecture which has implemented previously.

2.4 Benchmark

Currently, there is no existing model for this topic. A random guess model will be used. In the random guess model, the accuracy is roughly 1 in 15k since there are 15k classes. However, this dataset is not balance. If we chose the most frequent class in this dataset for every test, the result will be 4.1% accuracy (50337/1225029). Also, the precession of this model is 0.041.

$$= \frac{}{+} = \frac{50337}{50337 + 1174692} = 0.041$$

Therefore, if one example image contains one landmark (M = 1) and this model predicts the landmark correctly with one solution (P(1) = 0.041, rel(1) = 1), the GAP will be 0.041.

$$GAP = \frac{1}{M}\sum_{i=1}^{N} P(i)rel(i) = \frac{1}{1}\sum_{i=1}^{1} P(i)rel(i) = P(1)rel(1) = 0.041$$

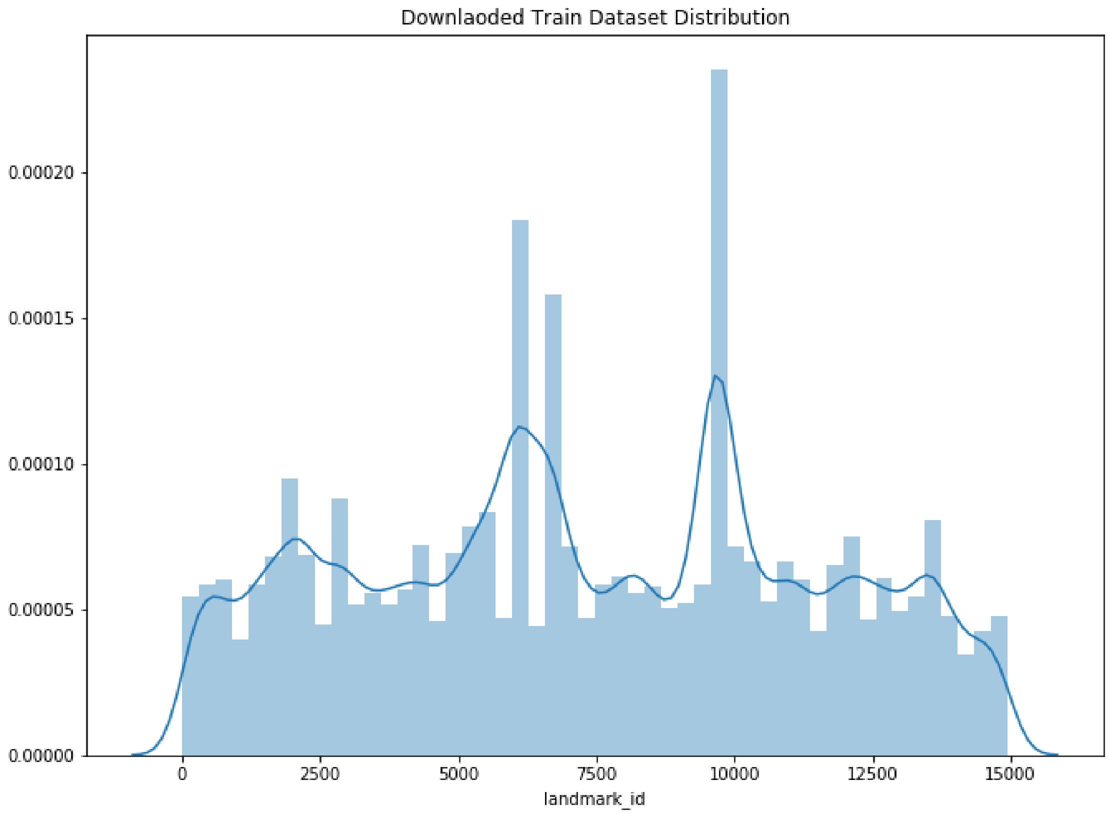
```
In [9]: # Get the landmark frequent of downloaded images.
        train_bm = pd.DataFrame(train.landmark_id.value_counts())
        train_bm.reset_index(inplace=True)
        train_bm.columns = ['landmark_id','count']
        display(train_bm.head())
        display(train_bm.tail())
        display(train_bm.shape)
        plt.figure(figsize = (11,8))
        plt.title('Downlaoded Train Dataset Distribution')
        sns.distplot(train['landmark_id'])
        plt.show()
```

```
   landmark_id  count
0         9633  24322
1         6051  24167
2         6599  11235
3         9779   9092
4         2061   6334
```

```
       landmark_id  count
14558         3005      1
14559          956      1
14560         9012      1
14561          208      1
14562         2129      1
```

```
(14563, 2)
```

```
C:\Users\Gamer\Anaconda3\envs\tfgpu\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarni
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



According to the above plot, the landmark 9633 has the most frequent. If an algorithm always recgonize an image with id 9633, the accuracy will be close to 4%.

```
In [10]: # If the algorithm predicts the image correctly, add one to bm_correct
         bm_correct = 0
         # The lankmark_id which the algorithm will predict
         bm_guess = 9633

         for landmark_id in train['landmark_id']:
             # always predict the image with the same landmark_id
             if landmark_id == 9633:
                 bm_correct += 1
         print('Accuracy: {x} %'.format(x=100*bm_correct/len(train['landmark_id'])))
```

```
Accuracy: 4.098994716574116 %
```

The above algorithm always has the same prediction, and the accuracy is 4.09899% which is close to 4%.

3 Methodology

3.1 Data Preprocessing

To avoid creating the relationship between the train dataset and test dataset, the test dataset will be only used for testing. The train dataset will be separated in two secitons, the training dataset and the valid dataset.

```
In [11]: train_rows = pd.DataFrame()
         valid_rows = pd.DataFrame()

         landmarks = pd.DataFrame(train['landmark_id'])
         landmarks = landmarks.drop_duplicates()
         display(landmarks.head())
         print(landmarks.shape)
         for landmark_id in tqdm(landmarks['landmark_id']):
             rows = train[ train['landmark_id'] == landmark_id]
             if len(rows.index) > 100:
                 samples = rows.sample(n=100)
                 rest = rows.drop(samples.index)
                 train_rows = train_rows.append(samples)
                 valid_rows = valid_rows.append(rest)
             elif len(rows.index) == 1:
                 train_rows = train_rows.append(rows)
             else:
                 samples = rows.sample(n=1)
                 rest = rows.drop(samples.index)
                 train_rows = train_rows.append(samples)
                 valid_rows = valid_rows.append(rest)

         display(train_rows.head())
         display(train_rows.shape)
         display(valid_rows.head())
         display(valid_rows.shape)
```

```
   landmark_id
0        10900
1         9633
2         7979
3         8487
4        10045
```

```
(14563, 1)
```

```
100%|| 14563/14563 [34:07<00:00,  7.11it/s]
```

```
                      id  landmark_id                                  filename
46171   f4d33bde35414fa9        10900  data/trainImages/f4d33bde35414fa9.jpg
295084  1c46a155a04383dc        10900  data/trainImages/1c46a155a04383dc.jpg
389777  95a12bfaab5d62ef        10900  data/trainImages/95a12bfaab5d62ef.jpg
215482  3873bebd998e66c2        10900  data/trainImages/3873bebd998e66c2.jpg
544913  80b25fd086f3893d        10900  data/trainImages/80b25fd086f3893d.jpg
```

```
(119899, 3)
```

```
                   id  landmark_id                                  filename
0    4cddf5dfec480378        10900  data/trainImages/4cddf5dfec480378.jpg
226  5f9ee6dad5124453        10900  data/trainImages/5f9ee6dad5124453.jpg
300  28575f73cf928ef2        10900  data/trainImages/28575f73cf928ef2.jpg
644  f5a9837932e8f121        10900  data/trainImages/f5a9837932e8f121.jpg
940  aef122f519be2d68        10900  data/trainImages/aef122f519be2d68.jpg
```

```
(473466, 3)
```

For the convenience, both data frame are stored. They can be loaded in the future.

```
In [14]: train_rows.to_csv('data/train_rows.csv', sep=',', index=False)
         valid_rows.to_csv('data/valid_rows.csv', sep=',', index=False)
```

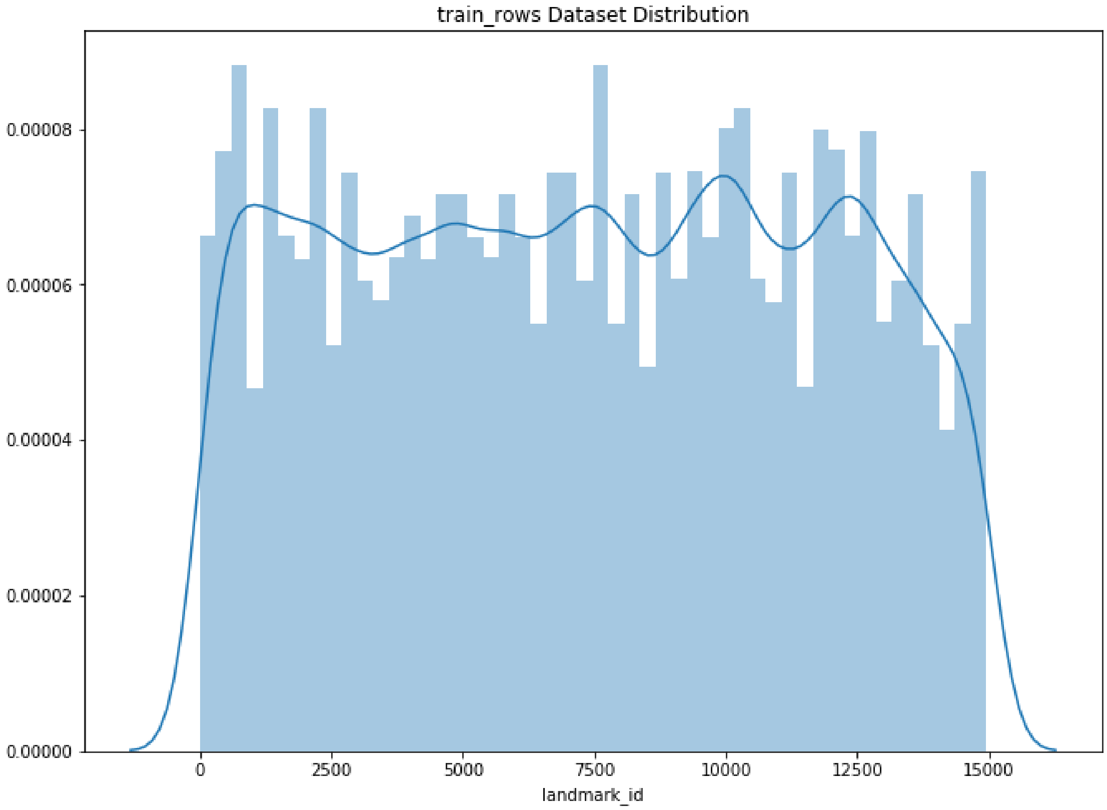
```
In [2]: train_rows = pd.read_csv('data/train_rows.csv')
        valid_rows = pd.read_csv('data/valid_rows.csv')
```

Let's check if the random selected test rows is balanced.

```
In [12]: train_rows_stat = pd.DataFrame(train_rows.landmark_id.value_counts())
         train_rows_stat.reset_index(inplace=True)
         train_rows_stat.columns = ['landmark_id','count']
         display(train_rows_stat.head())
         display(train_rows_stat.tail())

         plt.figure(figsize = (11,8))
         plt.title('train_rows Dataset Distribution')
         sns.distplot(train_rows['landmark_id'])
         plt.show()
         # plt.figure(figsize = (11,8))
         # plt.title('train_rows Dataset Distribution')
         # sns.distplot(valid_rows['landmark_id'])
         # plt.show()
```

```
   landmark_id  count
0        11423    100
```

```
1         10028    100
2          9155    100
3          5479    100
4          3788    100


       landmark_id  count
14558         9837      1
14559        11884      1
14560         7786      1
14561         3688      1
14562            0      1


C:\Users\Gamer\Anaconda3\envs\tfgpu\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarni
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

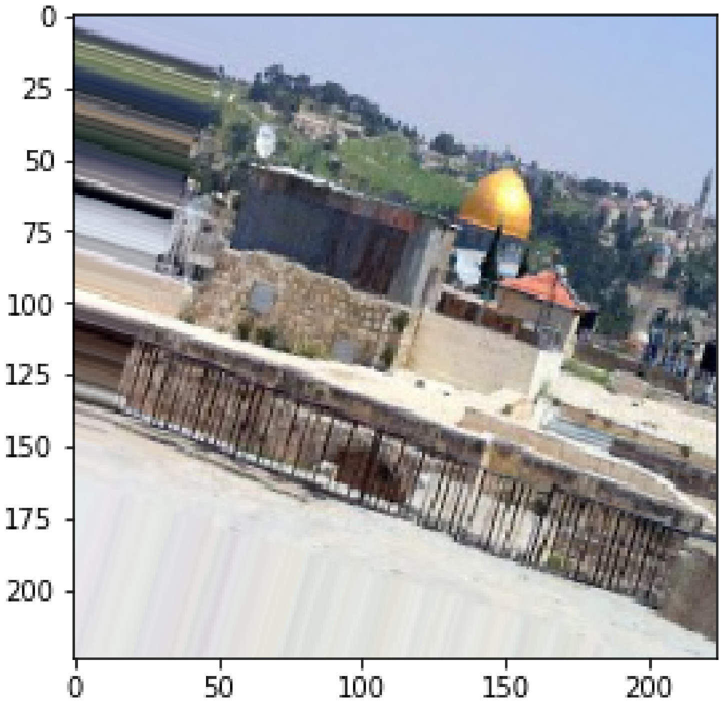


The train dataset is not balanced. There are several landmarks containing less than 100 images. The following will apply an image generater to balance train dataset.

```
In [3]: from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, l
In [14]: datagen = ImageDataGenerator(rotation_range=30.0,
                                      width_shift_range=0.2,
                                      height_shift_range=0.2,
                                      brightness_range=None,
                                      shear_range=0.0,
                                      zoom_range=0.0,
                                      fill_mode='nearest',
                                      horizontal_flip=True,
                                      rescale=None)

         def imgGenerator(file_path, nums, output_folder, showImage=False):
             '''
                 Image generator function
                 Argument:
                     file_path - the image file path which will be used to generate new images
                     nums - how many number of images will be generated
                     output_folder - the folder which will contain the agument image
                     showImage - if True, display the agument image
             '''
             resultImageList = []
             newImages = pd.DataFrame()
             previewImg = load_img(file_path)
             inputImg = img_to_array(previewImg)
             inputImg = inputImg.reshape((1,) + inputImg.shape)
             i = 0
             columns = 5
             # get the out file prefix based on the input file name.
             file_prefix = file_path.split('/')[-1].split('.')[0]
             # print(file_prefix)
             # image display initializer
             dataFig = plt.figure(figsize=(25,25))
             for batch in datagen.flow(inputImg, batch_size=1):
                 # Check if the file exists.
                 fileName = "{z}/{x}_{y}.jpg".format(z=output_folder,x=file_prefix, y=i)
                 # append to the image information list
                 resultImageList.append(fileName)
                 # check if the file exists
                 if os.path.exists(fileName):
                     i = i + 1
                     if i >= nums:
                         break
                     continue
                 try:
                     # convert to image file
                     imageToSave = array_to_img(batch[0])
                     # save the agument image
                     imageToSave.save(fileName, format='JPEG', quality=90)
                     # if True, show generated images.
```

```
        except:
            print("cannot save image: {x}.".format(x=file_prefix))
            i=i+1
        if showImage:
            plt.subplot(nums / columns + 1, columns, i+1)
            plt.imshow(imageToSave)

        i = i + 1
        if i >= nums:
            break
    return resultImageList

# Take one example to show how the imaGenerator function creates new images.
images_new_list = imgGenerator(file_path = 'data/trainImages/2e59d84de48709dd.jpg',
             nums = 1,
             output_folder = 'data',
             showImage = True)
print(images_new_list)
```

```
['data/2e59d84de48709dd_0.jpg']
```



In order to generate augment images, the following function is applied. The following cell will save all augment images and create a csv file recording images information including the file path and landmark id.

```
In [ ]: # use image generator to increase train dataset image numbers
        train_agument = pd.DataFrame(columns=['id','landmark_id','filename'])
        # Iterator train dataset with landmark id
        for t_row in tqdm(train_rows_stat.iterrows(), total=train_rows_stat.shape[0]):
            currnet_img_count = t_row[1]['count']
            # if the train dataset contrains 100 images, do not genrate new images
            if currnet_img_count == 100:
                continue
            # if the image count less than 100, generate new images.
            # how many images need to be generated
            img_augment_num = 100 - currnet_img_count
            # how many image can be generated by each existing image
            img_augment_rate = math.ceil(img_augment_num/currnet_img_count)
            # get the list of existing images.
            img_original = train_rows[train_rows['landmark_id'] == t_row[1]['landmark_id']]
            # iterator all existing images, and each image will generate the number of img_aug
            for i in range(img_original.shape[0]):
                images_new_list = imgGenerator(file_path = img_original.iloc[i]['filename'],
                         nums = img_augment_rate,
                         output_folder = 'data/trainImagesExtend',
                         showImage = False)
                # append the generated images information to a data frame.
                for filename in images_new_list:
                    # Check if the data set contaions 100 image
                    if img_augment_num <= 0:
                        break

                    img_id = filename.split('/')[-1].split('.')[0]
                    train_agument.loc[train_agument.shape[0]] = [img_id, t_row[1]['landmark_id
                    # update the rest of image will be generated
                    img_augment_num = img_augment_num -1

        display(train_agument.head())
        display(train_agument.shape)
        train_agument.to_csv('data/train_agument.csv', sep=',', index=False)
```

```
In [4]: train_agument = pd.read_csv('data/train_agument.csv')
```

After generating agumnet images, check if the train dataset is balance or not.

```
In [5]: train_input = train_rows.append(train_agument)
        train_input_stat = pd.DataFrame(train_input.landmark_id.value_counts())
        train_input_stat.reset_index(inplace=True)
        train_input_stat.columns = ['landmark_id','count']
        display(train_input_stat.head())
        display(train_input_stat.tail())

        # plt.figure(figsize = (11,8))
```