In this assignment, I have implemented 3 types of instance search methods which are **CNN**, **Color Histogram** and **LBP + Color Histogram**. And based on my testing, the performance can be rank by **CNN > Color Histogram > LBP + Color Histogram**. In the following parts, I will briefly describe the methods and list the top 10 search results for every 20 query instances using CNN.

Image Preprocessing

Before applying the algorithm, I first use 5 hours to cropping of 5000 gallery images. Here I use a tool called labelImg (https://github.com/heartexlabs/labelImg) label the object location of each image. And labeled image will output a .xml file saved at '\Image Data\Dataset Image\gallery_crop_info' file. Next, I created a program '\Image Data\Dataset Image\get_crop_txt.py' to convert all .xml files to .txt files and saved the format as the same as example query .txt file at '\Image Data\Dataset Image\gallery_txt_4186' file. Finally, run the '\Image Data\crop_img.py' get the cropped image of gallery dataset. All of the cropped image are saved at '\Image Data\Dataset Image\gallery_croped_4186'

CNN

For the CNN method, I use Xception model to extract the image feature for 2 different model layers avg and max base on image size '299 X 299'. The extraction code and ranking code are located at '\Implementation\Method1_CNN\cnnExtraction.py' and '\Implementation\Method1_CNN\cnnRankList.py'. Below is the step of implementations:

Extraction:

1. Define the gallery, query image part and the feature save path of avg and max
2. Create two Xception model layers of avg and max using tensorflow API
3. Chane the image to RGB format and resize it to '299 X 299'
4. Pass preprocessed image to Xception model avg, max layer and saved the feature as .npy file at '\Implementation\Method1_CNN\feature'
5. Repeat the step until finished all query and gallery image

Ranking:

The distance measuring, I use cosine similarity to help me get the .txt ranking result and saved at '\Implementation\Method1_CNN/rank_list.txt'.

Color Histogram

For the Color Histogram method is **implemented by myself without calling any library or API**, I use HSV + RGB color space to loop through each pixel value and normalize histogram values by dividing by total number of pixels. The extraction code and ranking code are located at '\Implementation\

Method2_ColorHistogram\color_histogram.py' and '\Implementation\ Method2_ColorHistogram\color_histogram_ranking.py'. Below is the step of implementations:

Extraction:

1. Define the gallery, query image part and the histogram data save path
2. Use opencv function convert BGR image to HSV and RGB
3. Use double for loop count each pixel of histogram
4. Normalize histogram values by dividing by total number of pixels
5. Repeat the step until finished all query and gallery image

Ranking:

The distance measuring, I use Euclidean distance to help me get the distance of HSV and RGB of .txt ranking result and saved at '\Implementation\Method2_ColorHistogram /rank_list.txt'.

LBP + Color Histogram

For the LBP + Color Histogram method is **implemented by myself without calling any library or API**, I first calculate the LBP pixel value of LBP image + HSV color space to loop through each pixel value of LBP image and normalize histogram values by dividing by total number of pixels. The extraction code and ranking code are located at '\Implementation\Method3_LBP_HSV\LBP.py', '\Implementation\Method3_LBP_HSV\color_histogram_HSV.py' and '\Implementation\Method3_LBP_HSV \LBP_color_his_ranking.py'. Below is the step of implementations:

RGB image of LBP image:

1. Define the gallery, query image part and the histogram data save path
2. Use opencv function convert BGR image to GRAY
3. Use double for loop count each pixel of LBP value
4. Saved the LBP image at '\Image Data\Dataset Image\gallery_croped_LBP' and '\Image Data\Testing Image\query_croped_LBP'
5. Repeat the step until finished all query and gallery image

Extraction:

1. Define the gallery, query image part and the histogram data save path
2. Use double for loop count each pixel of histogram
3. Normalize histogram values by dividing by total number of pixels
4. Repeat the step until finished all query and gallery image

Ranking:

The distance measuring, I use Euclidean distance to help me get the .txt ranking result and saved at '\Implementation\ Method3_LBP_HSV /rank_list.txt'.

Time Cost

Image preprocessing: 5 hours

CNN using GPU: 5~6 min

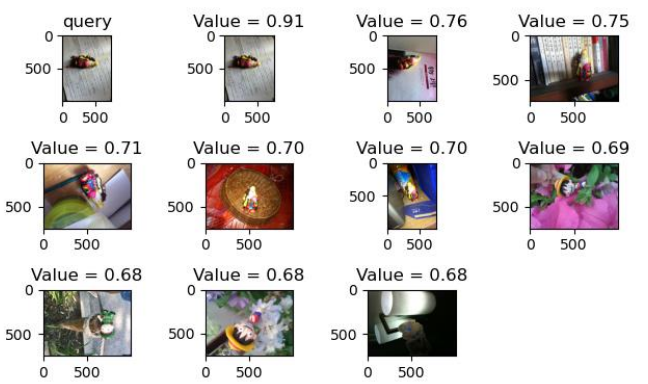
CNN without using GPU: 1~1.5 hour

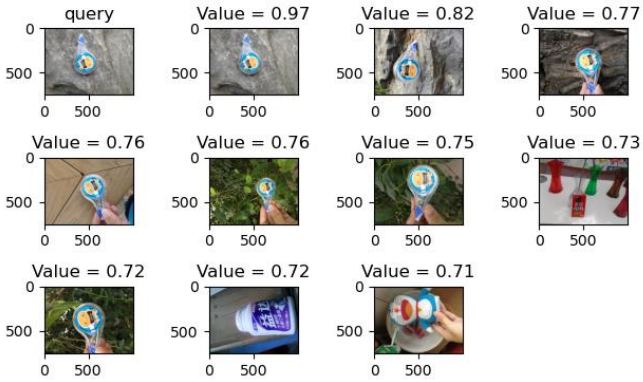
Color Histogram: 15 min

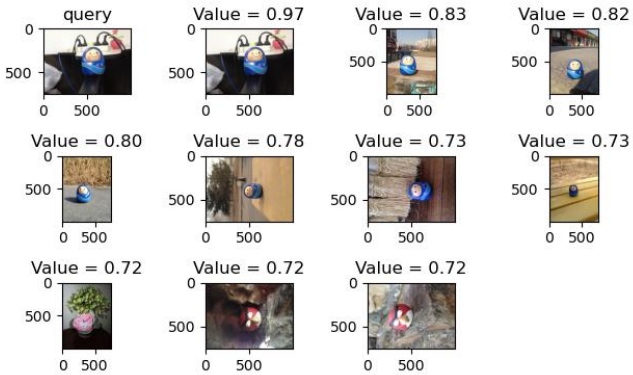
LBP + Color Histogram: 30 min

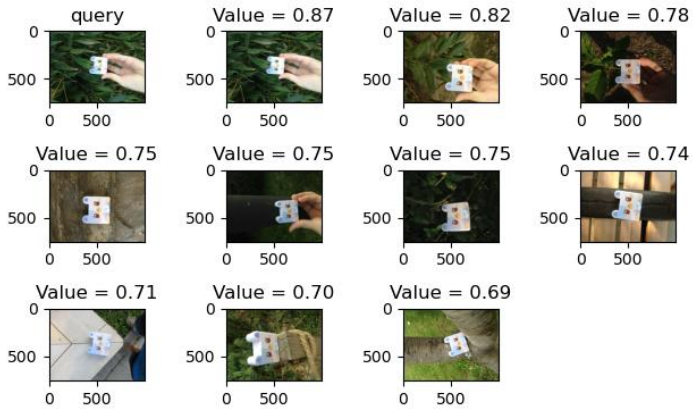
Conclusion

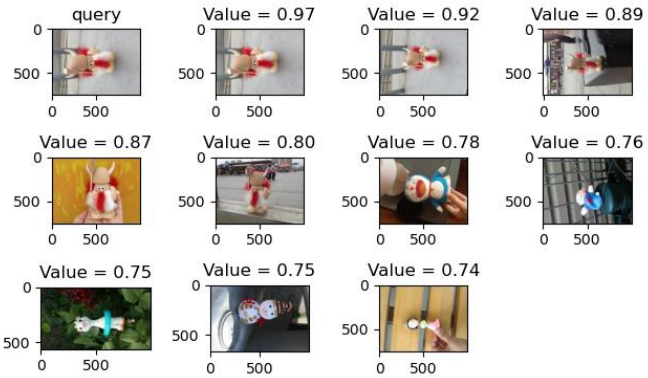
Consider the time and performance, I get the best method is CNN basically all of query image can get the true result, below shows the result of top 10 on 20 query image:

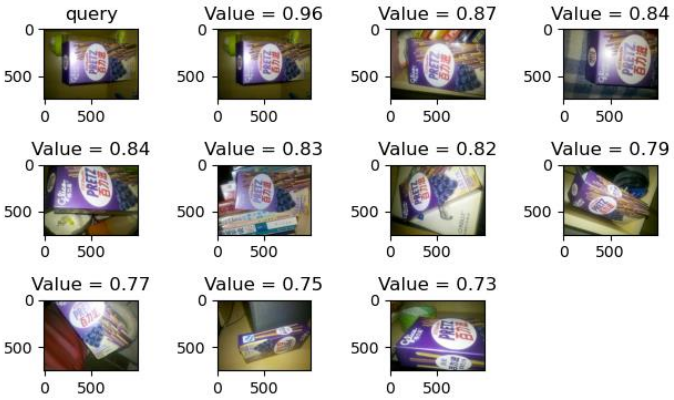


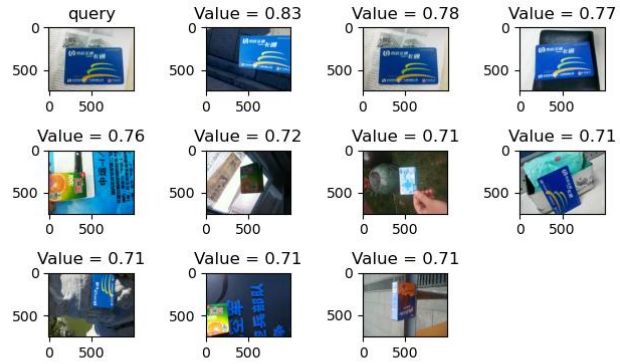


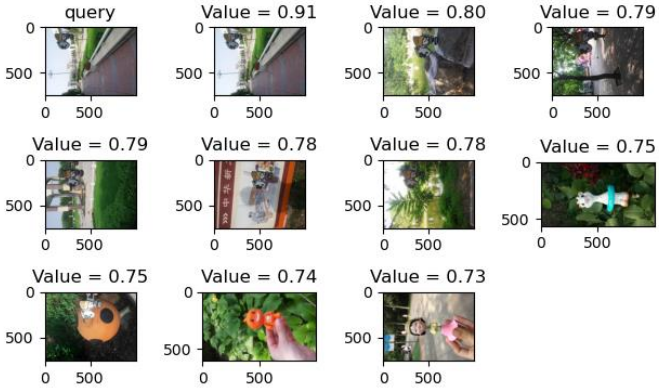


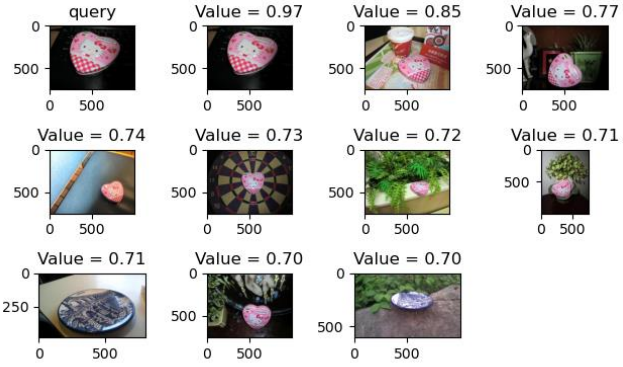


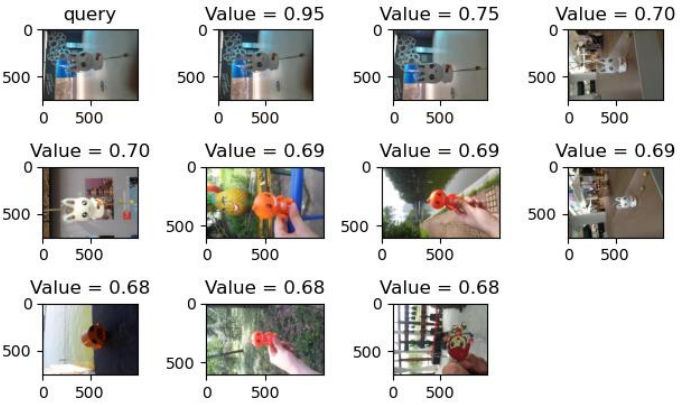

query
Value = 0.97
Value = 0.92
Value = 0.89
Value = 0.87
Value = 0.80
Value = 0.78
Value = 0.76
Value = 0.75
Value = 0.75
Value = 0.74

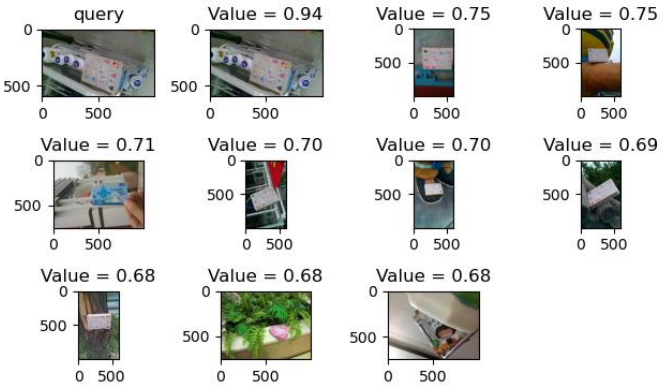

query
Value = 0.96
Value = 0.87
Value = 0.84
Value = 0.84
Value = 0.83
Value = 0.82
Value = 0.79
Value = 0.77
Value = 0.75
Value = 0.73

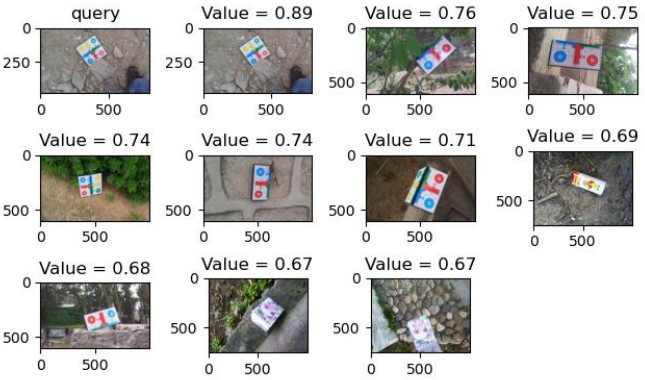

query
Value = 0.83
Value = 0.78
Value = 0.77
Value = 0.76
Value = 0.72
Value = 0.71
Value = 0.71
Value = 0.71
Value = 0.71
Value = 0.71

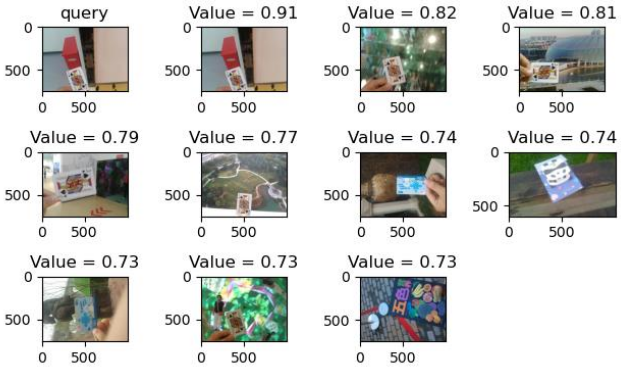

query
Value = 0.91
Value = 0.80
Value = 0.79
Value = 0.79
Value = 0.78
Value = 0.78
Value = 0.75
Value = 0.75
Value = 0.74
Value = 0.73

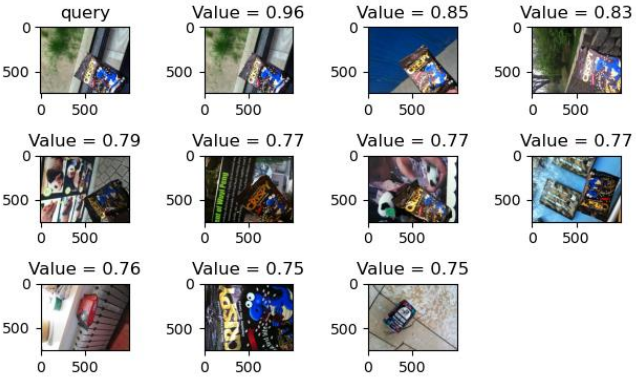

query
Value = 0.97
Value = 0.85
Value = 0.77
Value = 0.74
Value = 0.73
Value = 0.72
Value = 0.71
Value = 0.71
Value = 0.70
Value = 0.70

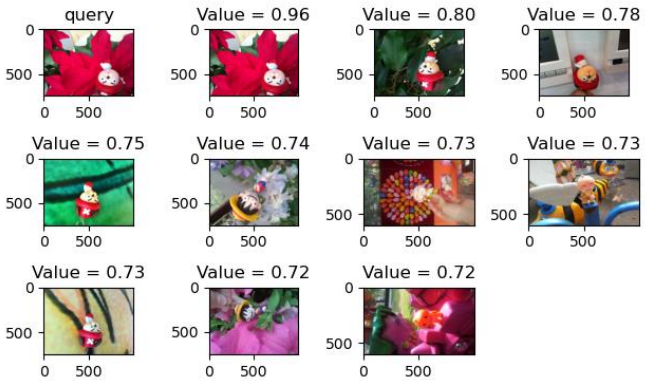

query
Value = 0.95
Value = 0.75
Value = 0.70
Value = 0.70
Value = 0.69
Value = 0.69
Value = 0.69
Value = 0.68
Value = 0.68
Value = 0.68

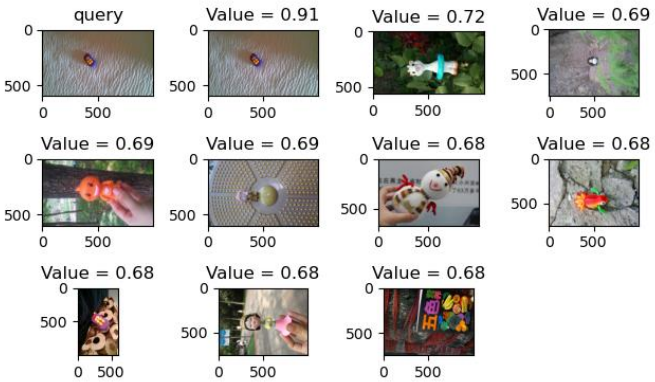

query
Value = 0.94
Value = 0.75
Value = 0.75
Value = 0.71
Value = 0.70
Value = 0.70
Value = 0.69
Value = 0.68
Value = 0.68
Value = 0.68

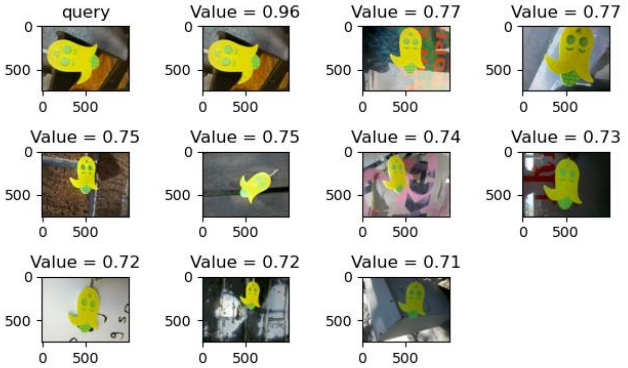

query
Value = 0.89
Value = 0.76
Value = 0.75
Value = 0.74
Value = 0.74
Value = 0.71
Value = 0.69
Value = 0.68
Value = 0.67
Value = 0.67

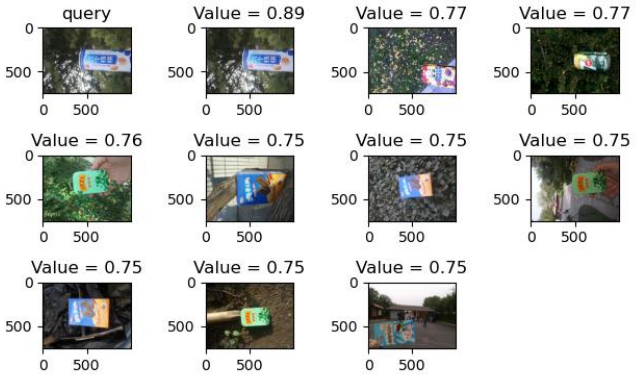

query
Value = 0.91
Value = 0.82
Value = 0.81
Value = 0.79
Value = 0.77
Value = 0.74
Value = 0.74
Value = 0.73
Value = 0.73
Value = 0.73

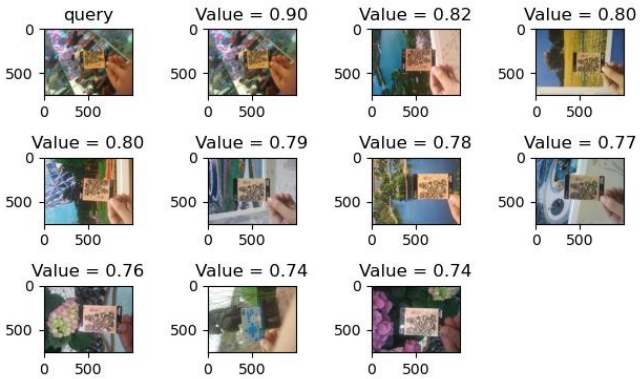

query
Value = 0.96
Value = 0.85
Value = 0.83
Value = 0.79
Value = 0.77
Value = 0.77
Value = 0.77
Value = 0.76
Value = 0.75
Value = 0.75


query
Value = 0.96
Value = 0.80
Value = 0.78
Value = 0.75
Value = 0.74
Value = 0.73
Value = 0.73
Value = 0.73
Value = 0.72
Value = 0.72


query
Value = 0.91
Value = 0.72
Value = 0.69
Value = 0.69
Value = 0.69
Value = 0.68
Value = 0.68
Value = 0.68
Value = 0.68
Value = 0.68


query
Value = 0.96
Value = 0.77
Value = 0.77
Value = 0.75
Value = 0.75
Value = 0.74
Value = 0.73
Value = 0.72
Value = 0.72
Value = 0.71


query
Value = 0.89
Value = 0.77
Value = 0.77
Value = 0.76
Value = 0.75
Value = 0.75
Value = 0.75
Value = 0.75
Value = 0.75
Value = 0.75


query
Value = 0.90
Value = 0.82
Value = 0.80
Value = 0.80
Value = 0.79
Value = 0.78
Value = 0.77
Value = 0.76
Value = 0.74
Value = 0.74