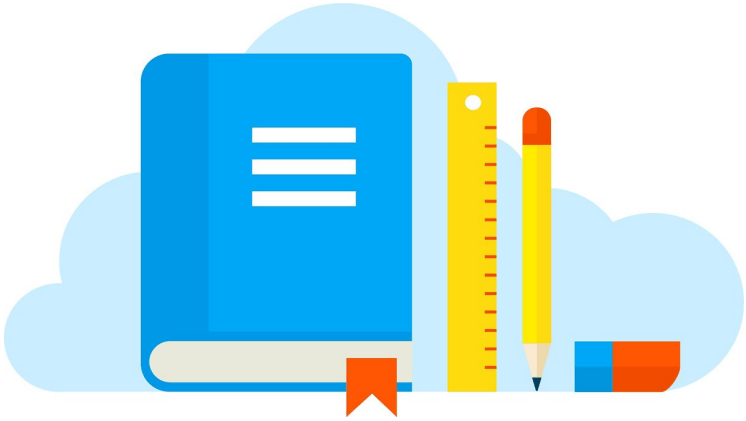


CSE 573

Programming Assignment 3

Image Based Search Engine

Vanshika Nigam
50208031

INTRODUCTION

An image based search engine is one that is designed to find an image. For this report a search engine designed in Python by Adrian Rosebrock is used to successfully index the images and then retrieve a relevant image on doing an image search. It is based on the explanation of an image search engine and its analysis using real world photos of different landmarks of the University at Buffalo. The landmarks include:

1. **Davis Hall**
2. **The Commons**
3. **Student Union**
4. **Capen Hall**
5. **Hayes Hall**

All the images were taken from a smartphone camera and later the resolution was downgraded to 400 x 166 as per the requirement of the code. The number of images taken per landmark were 6 of which 5 were indexed and 1 served as a query image. Thus a total of 30 images were clicked.

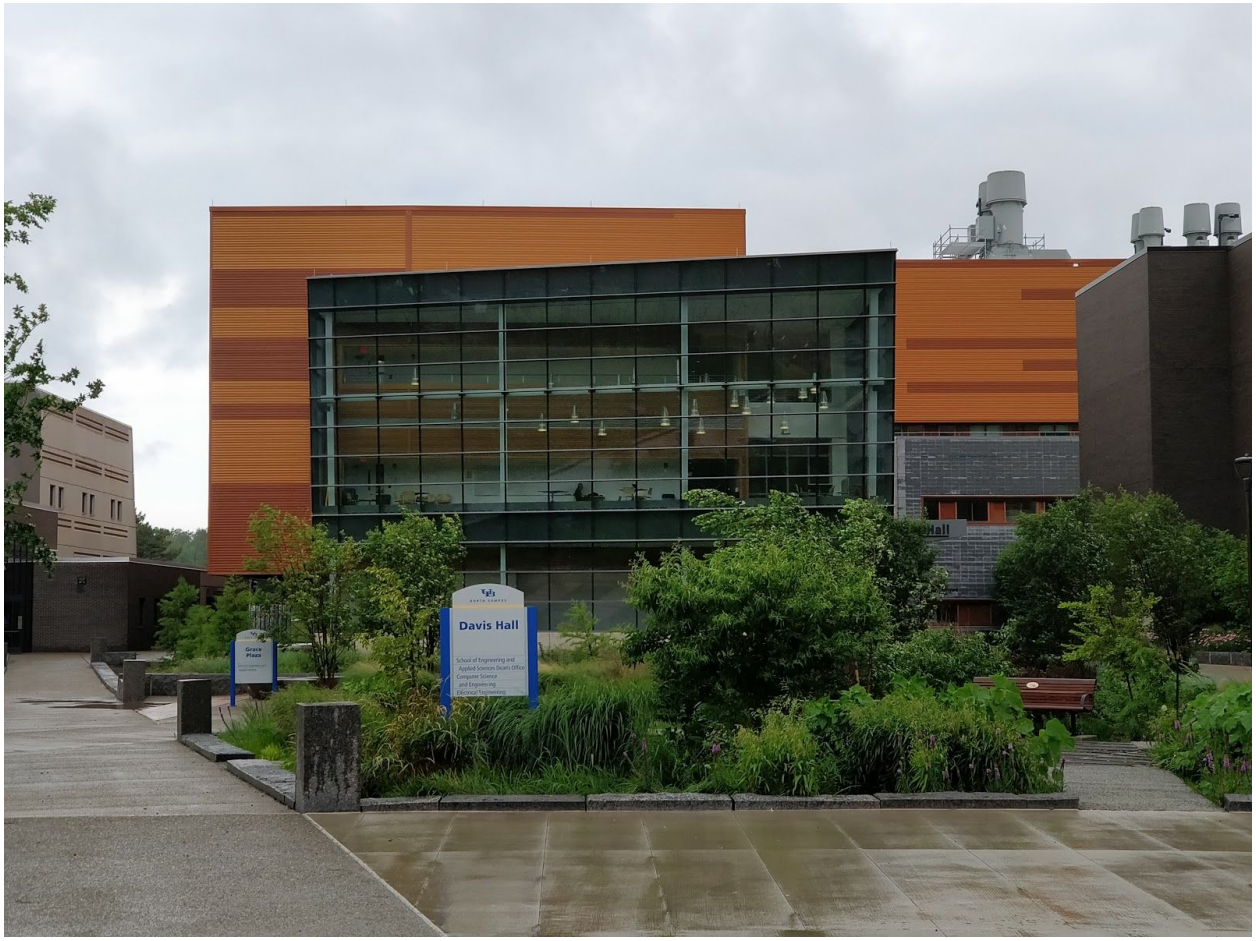
The first thing was to do is index the 25 images in the dataset. Indexing is the process of quantifying the dataset by using an image descriptor to extract features from each image and storing the resulting features for later use, such as performing a search. The indexing is done by the index.py script.

Now the script is given a query image and it ranks the images in the index based on how relevant they are to the fed query. Two feature vectors can be compared using a distance metric. A distance metric is used to determine how "similar" two images are by examining the distance between the two feature vectors.

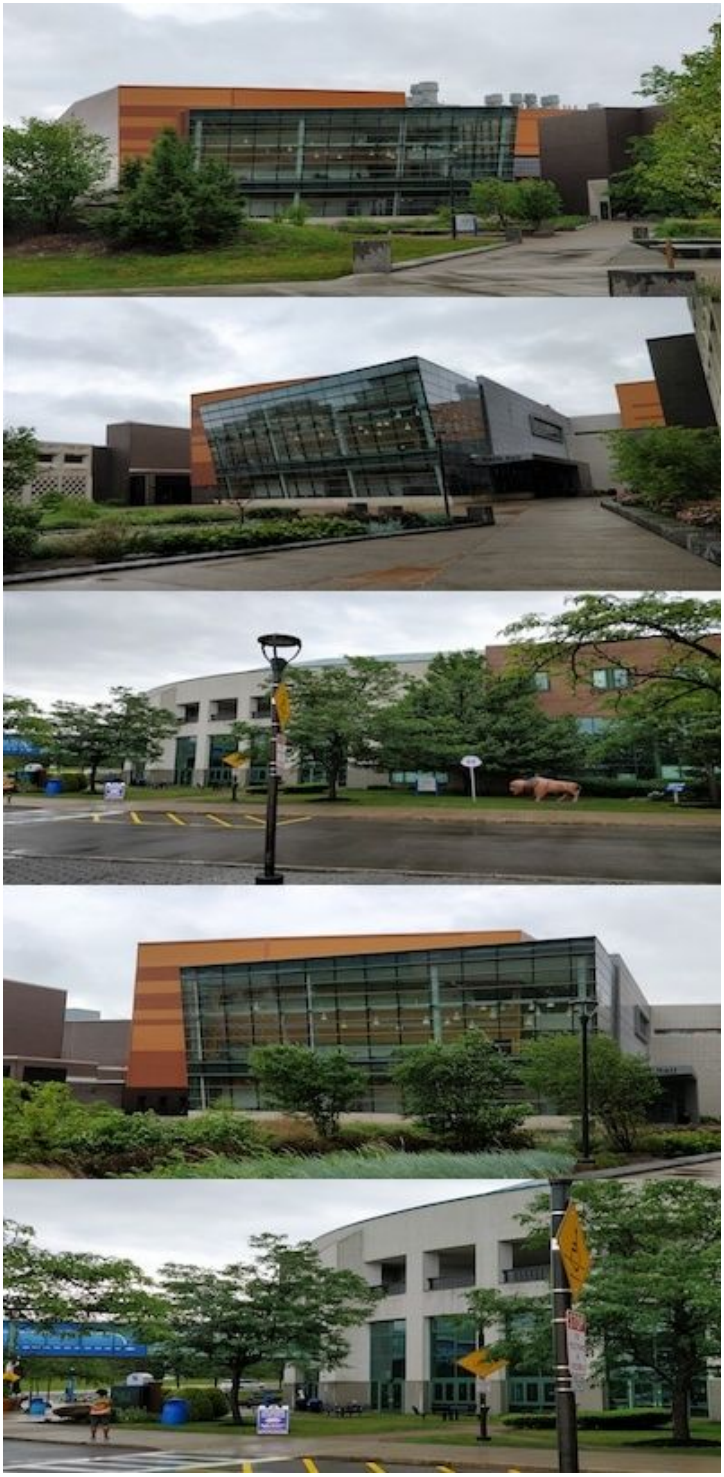
Thus if we present an image search engine with a query image, we expect it to return images that are relevant to the content of image — hence, we sometimes call image search engines by what they are more commonly known in academic circles as Content Based Image Retrieval (CBIR) systems.

QUERY RESULTS AND ANALYSIS

1. Query Image for Davis Hall :



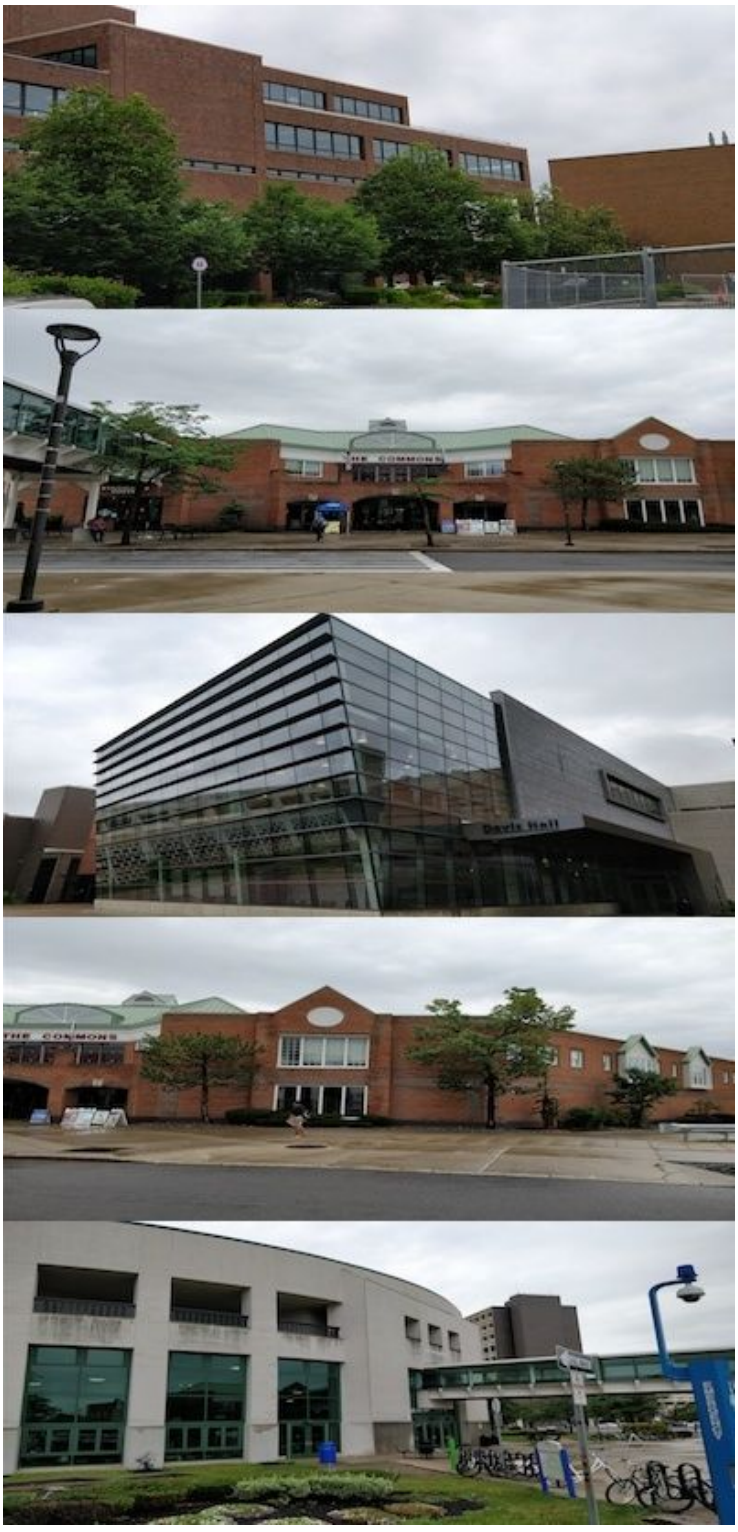
The Top 5 Search Results were :



Accuracy: 3/5

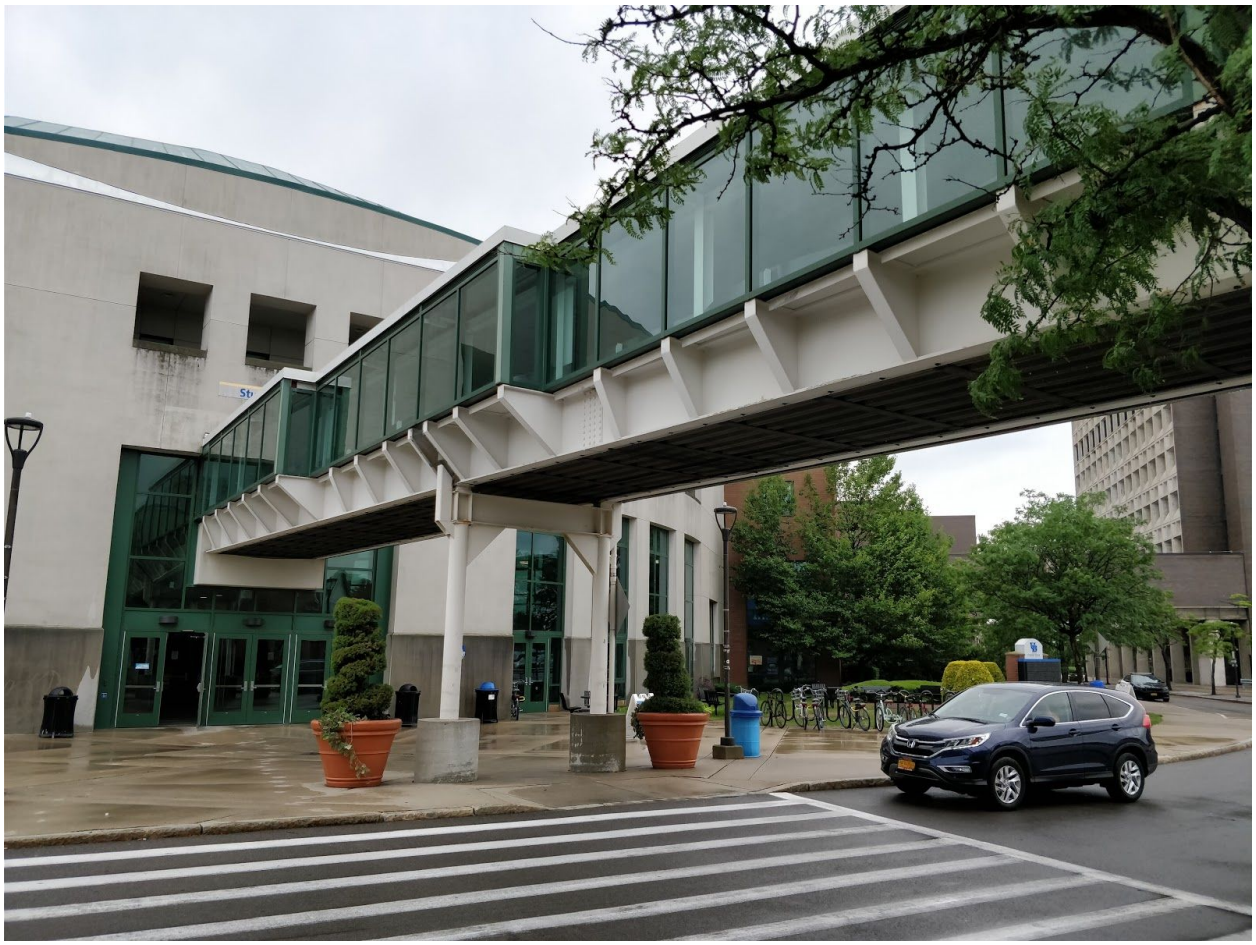
Accuracy Percentage : 60%

The next 5 results were ：

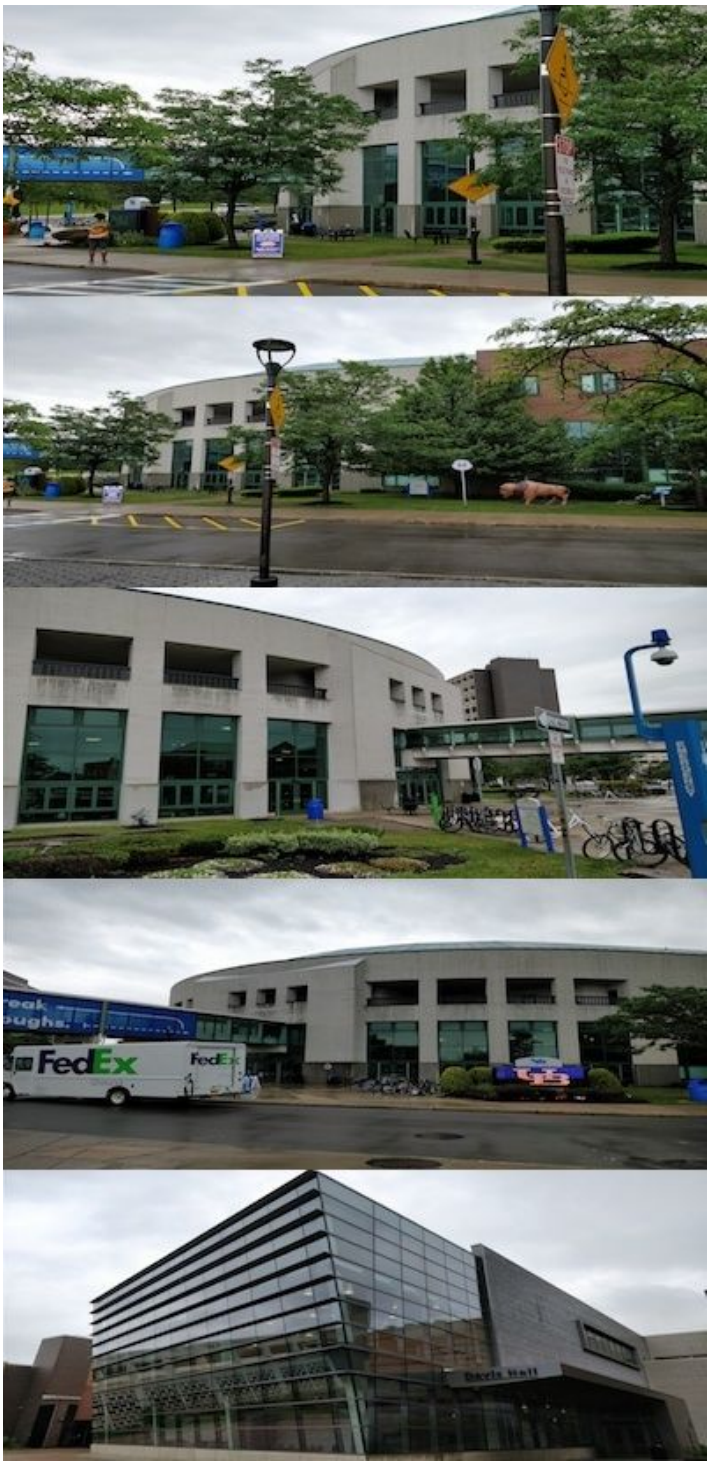


All the 5 results were not obtained in the top 5 section due to similarity with other images present in the dataset.

2. Query Image for **Student Union**:



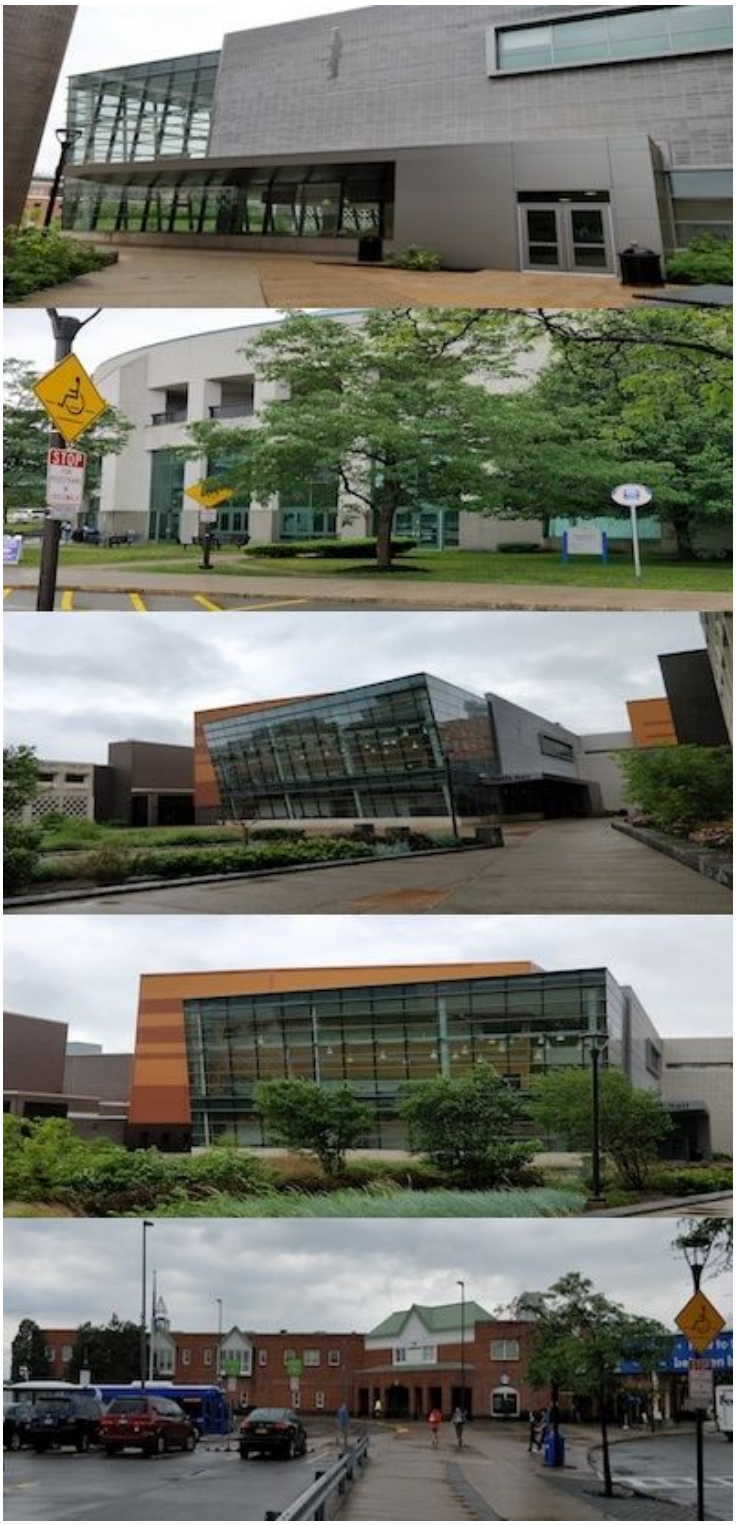
The Top 5 Search Results were :



Accuracy: 4/5

Accuracy Percentage : 80%

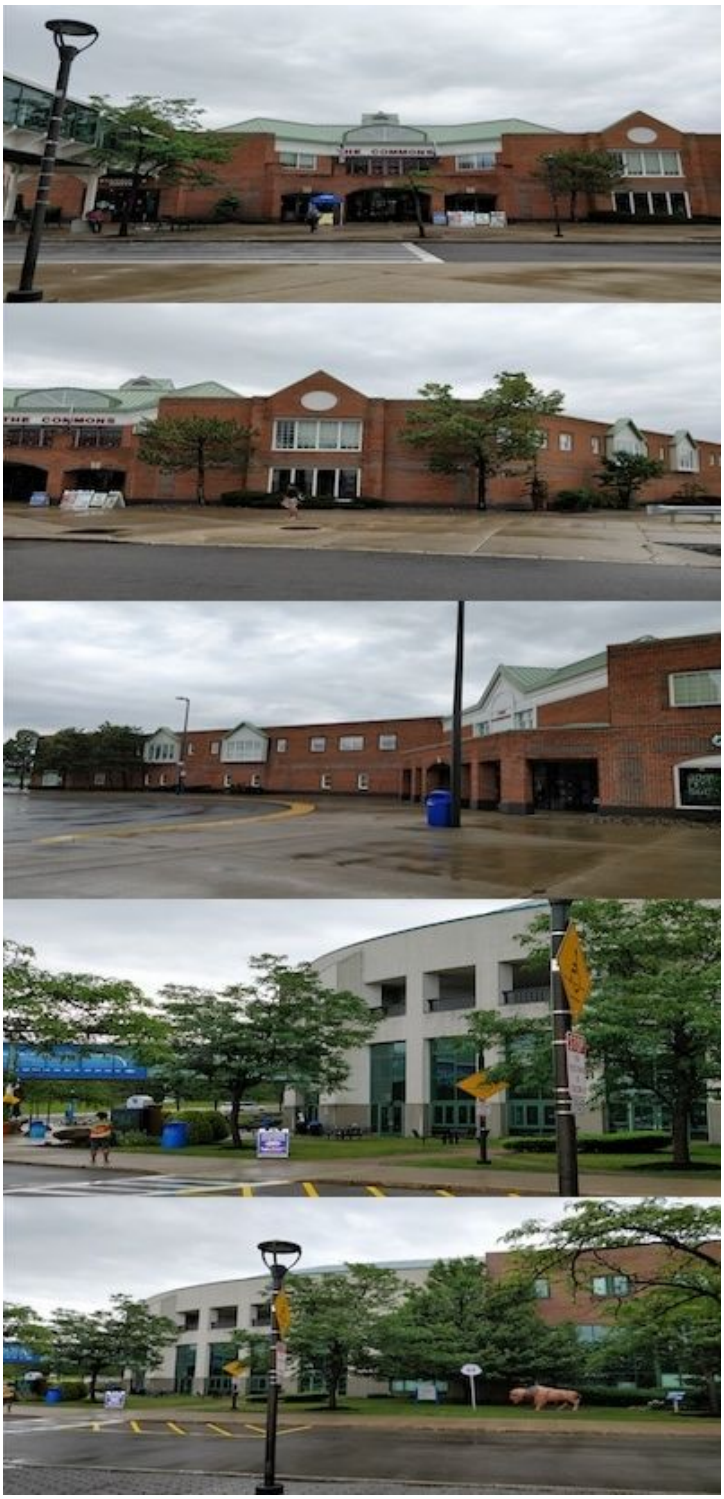
The next 5 results were ：



Only 4 results were not obtained in the top 5 section and 1 left was obtained in the next slot since the windows being of glass resembled other images present in the dataset.

3. Query Image for **The Commons** :

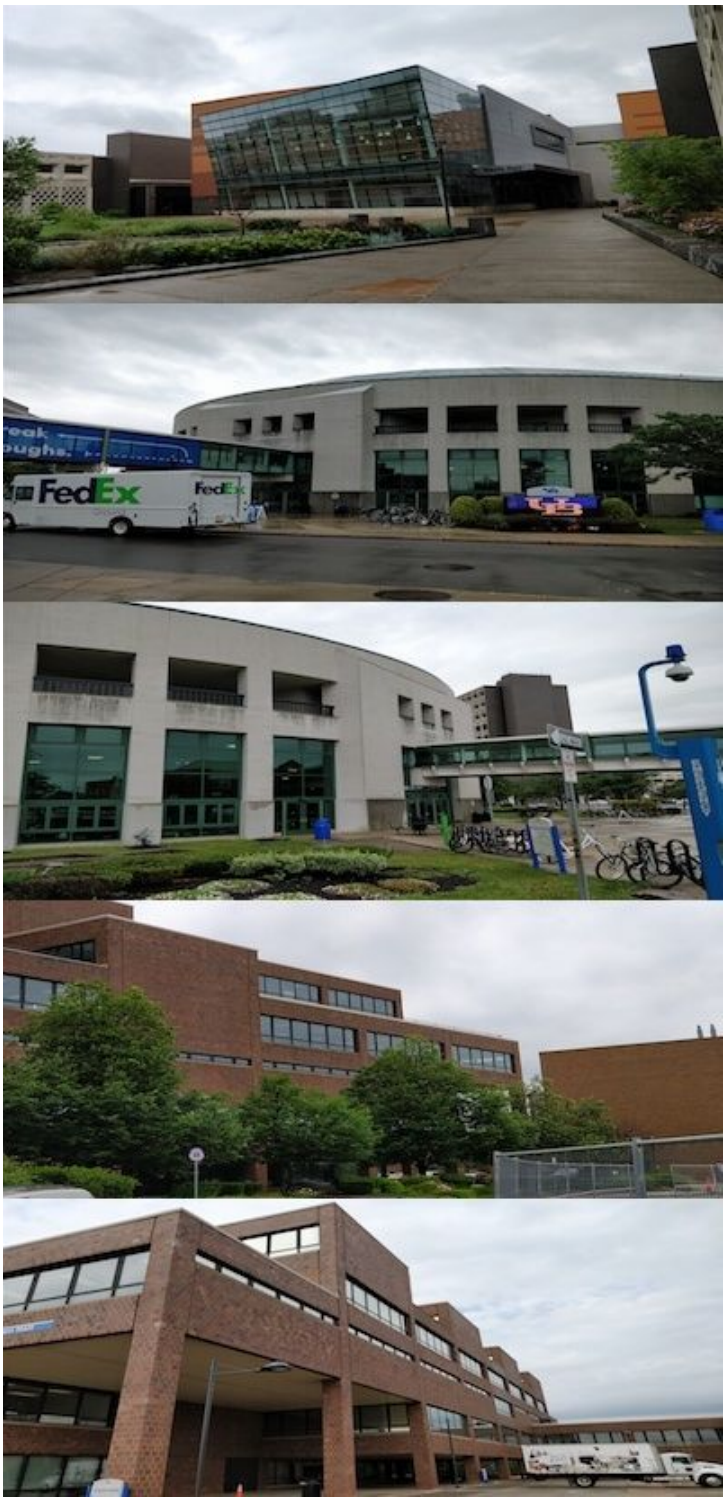
The Top 5 Search Results were :



Accuracy: 3/5

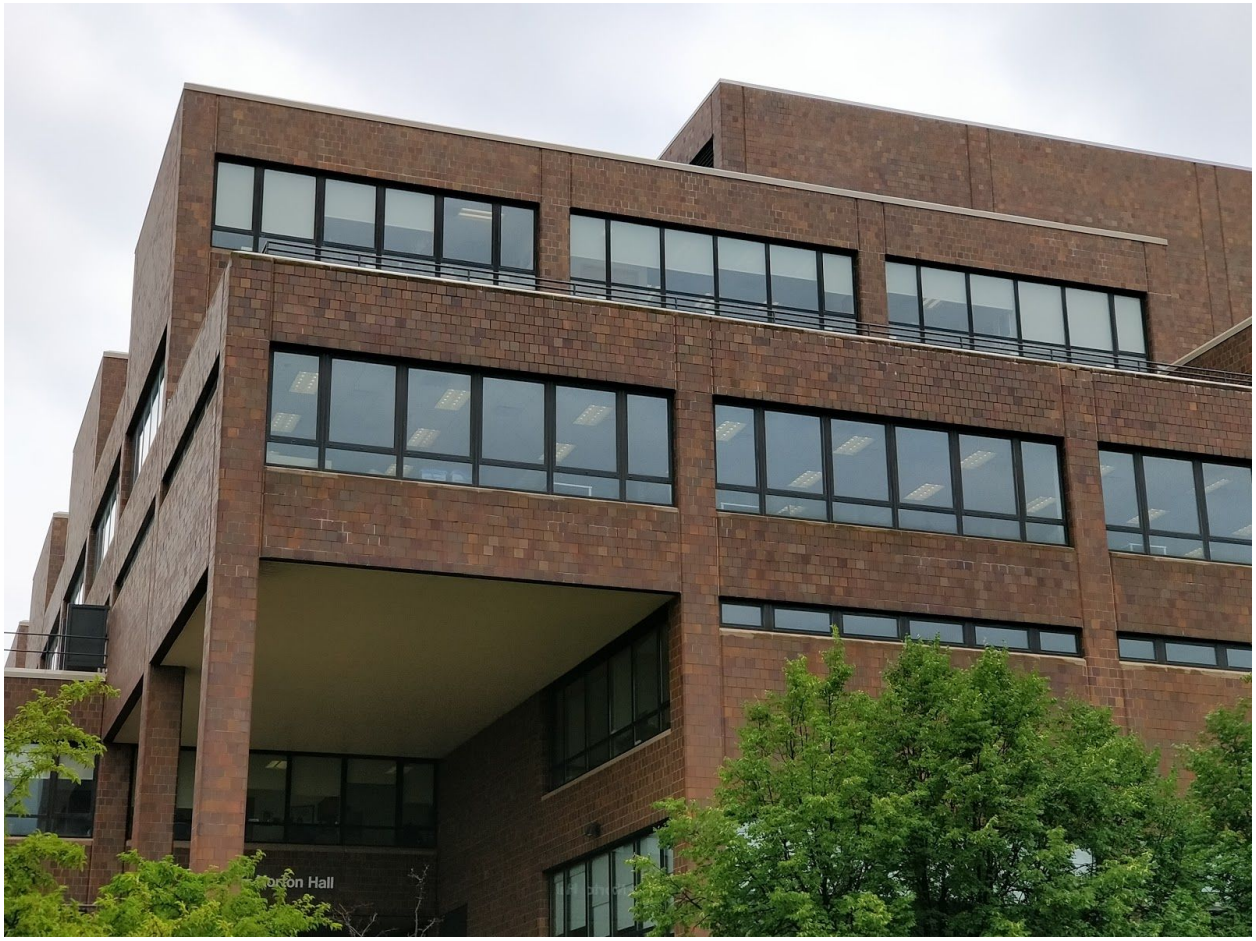
Accuracy Percentage : 60%

The next 5 results were ：

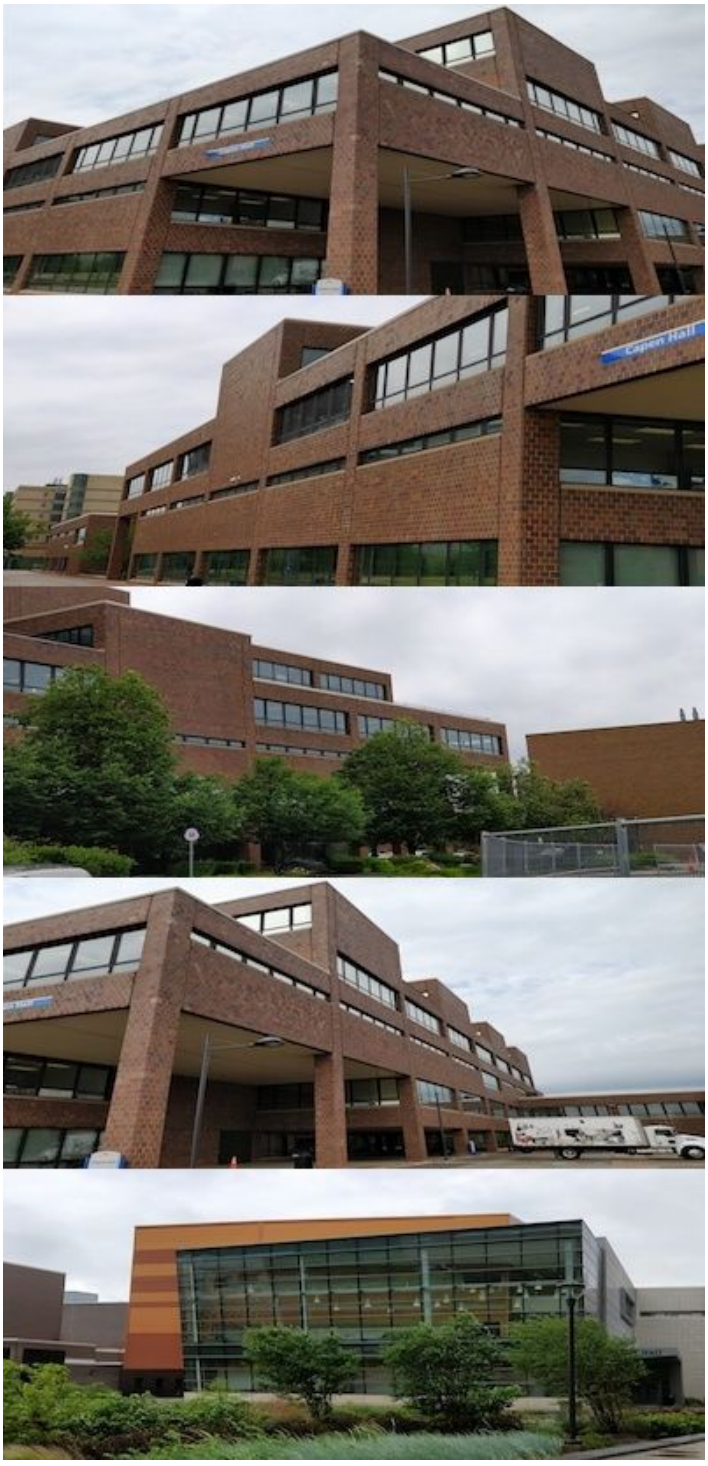


There is a high similarity between the buildings of Capen Hall and Commons because of the red brick structure. Thus the Capen Hall images are also present in the query result.

4. Query Image for **Capen Hall** :



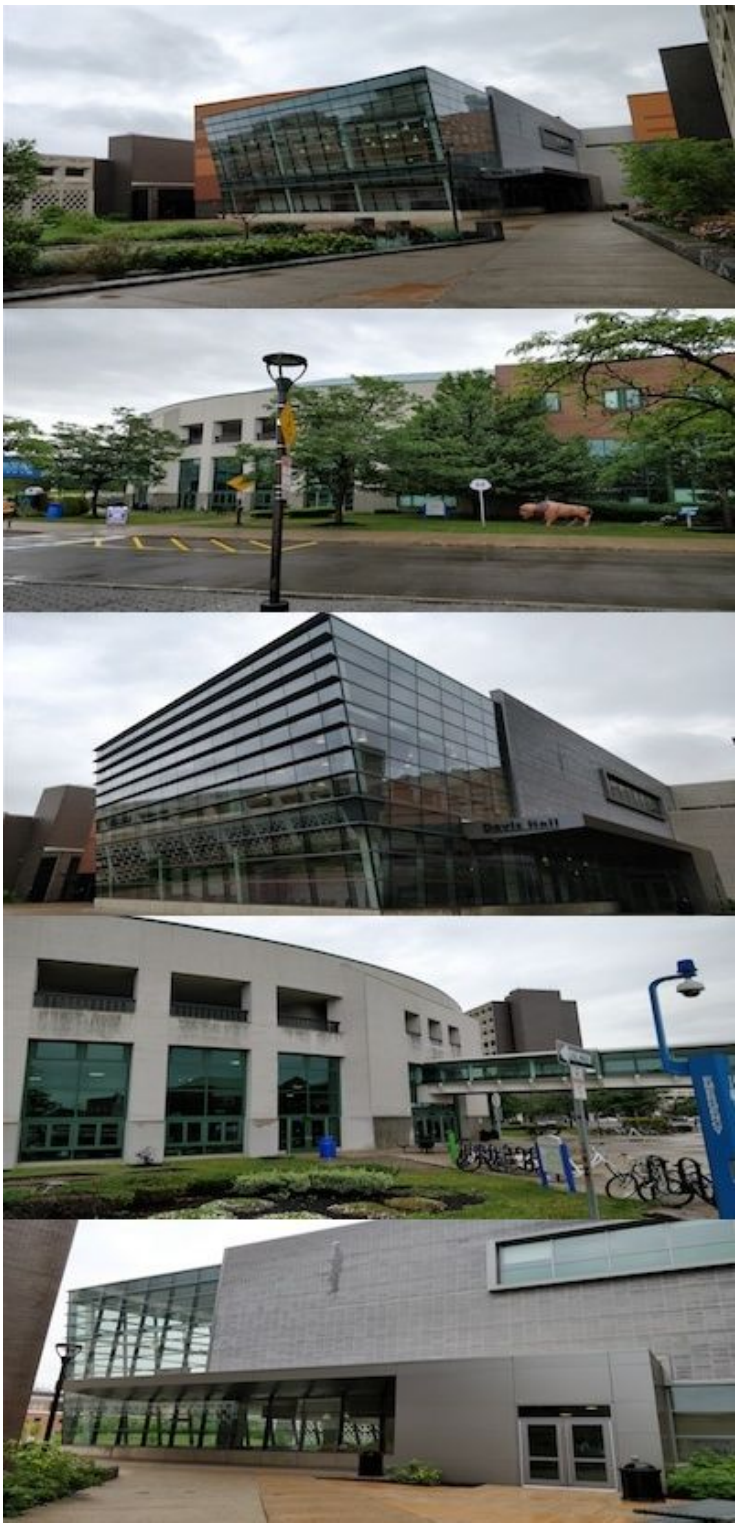
The Top 5 Search Results were :



Accuracy: 4/5

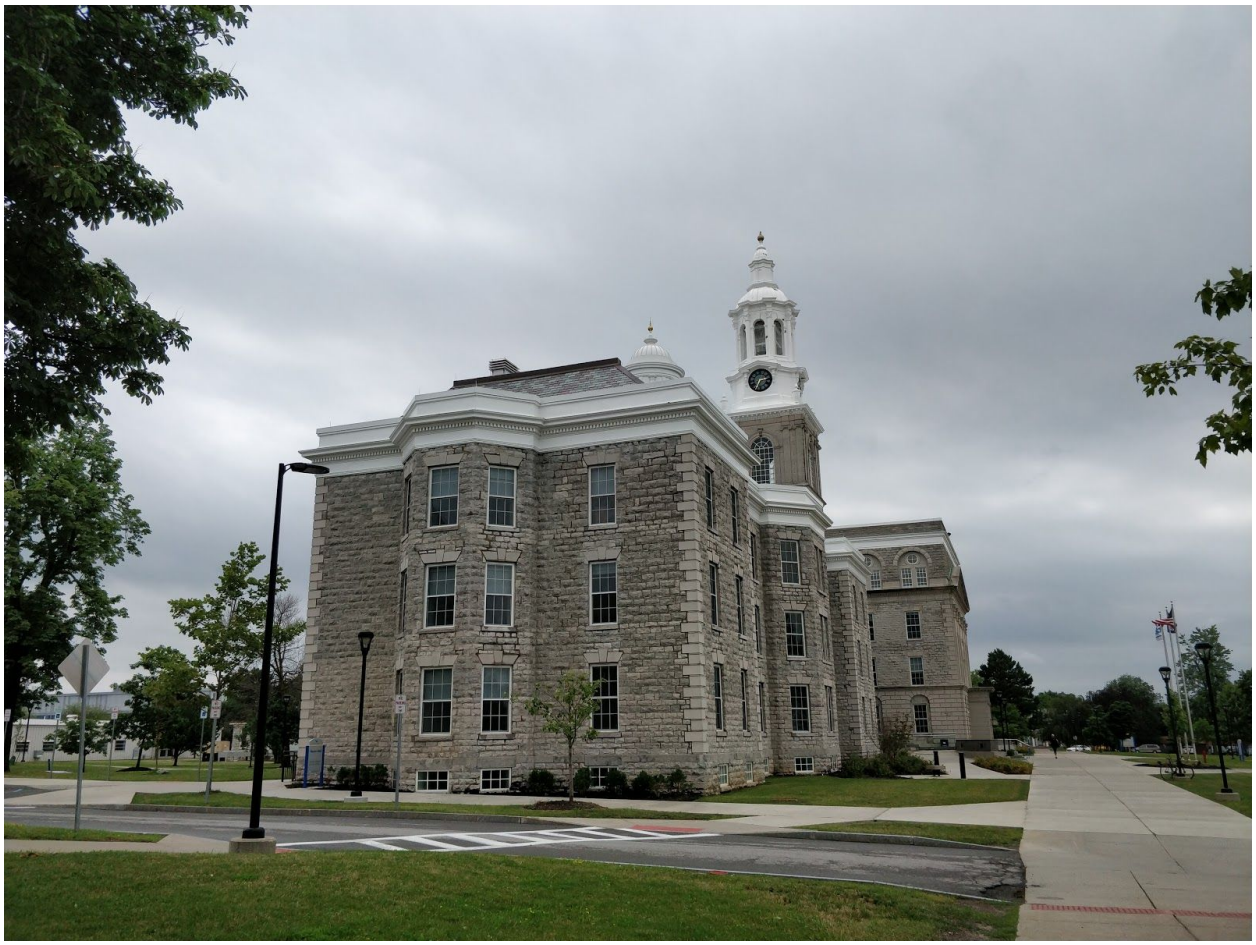
Accuracy Percentage : 80%

The next 5 results were ：

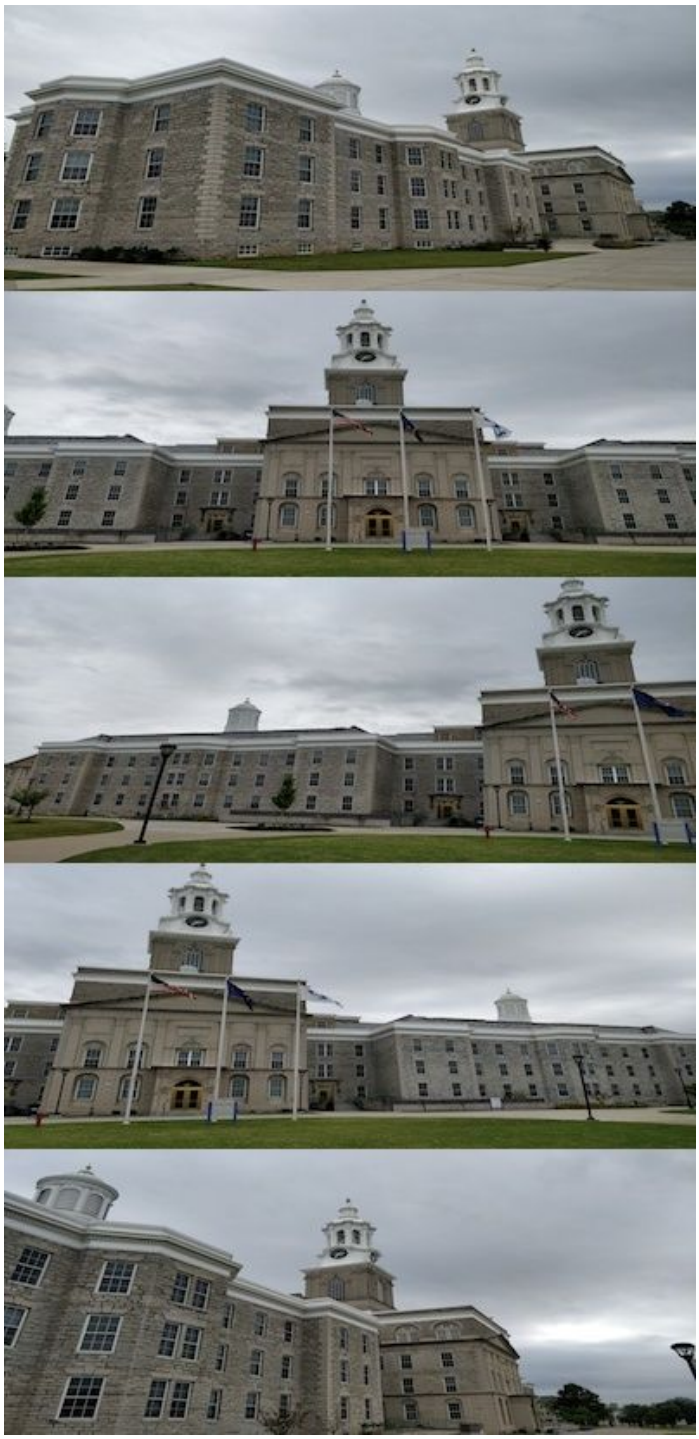


Not all 5 images were obtained in the top 5 section because of the similarity of backgrounds like the sky, greenery etc.

5. Query Image for **Hayes Hall** :



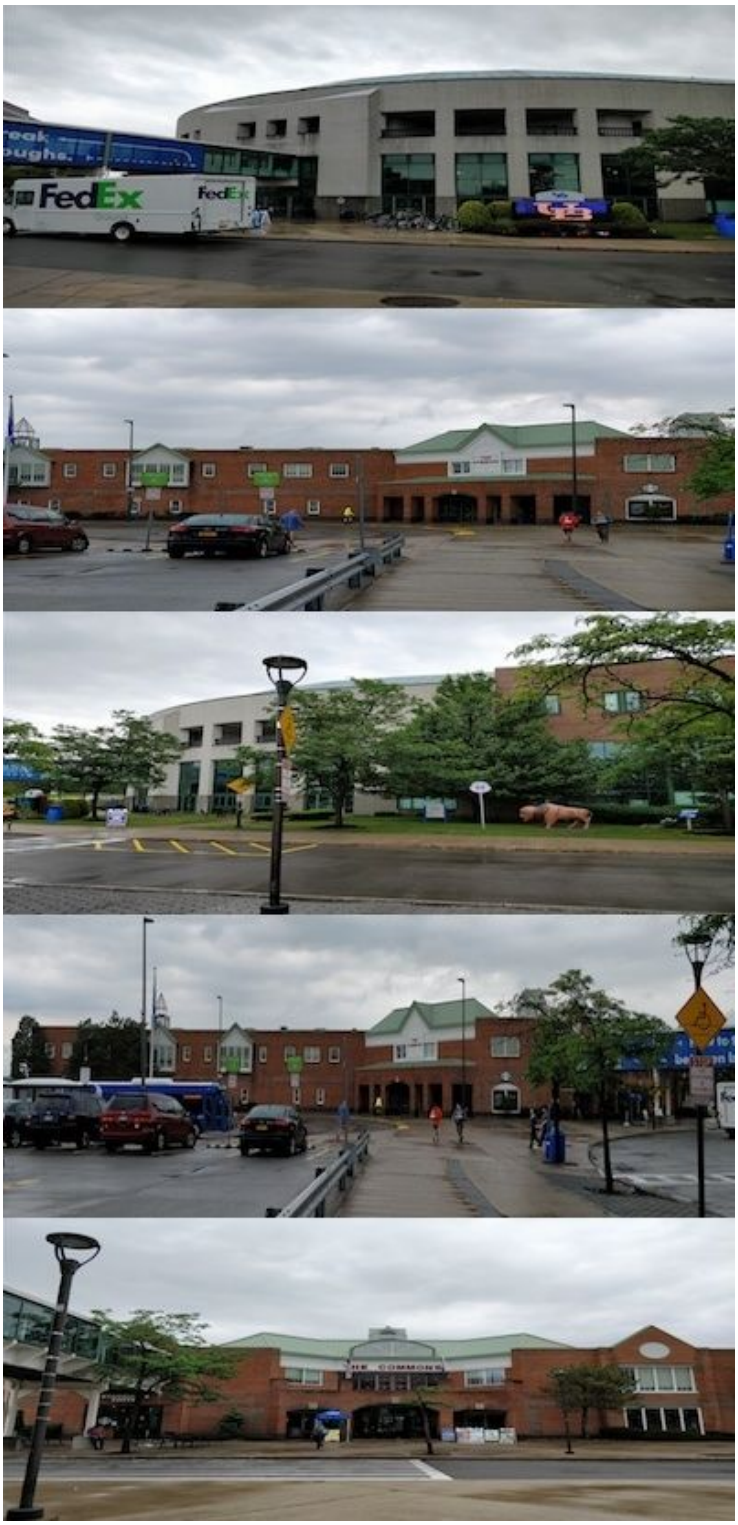
The Top 5 Search Results were :



Accuracy: 5/5

Accuracy Percentage : 100%

The next 5 results were ：



All the 5 images were obtained for Hayes Hall in the top 5 section due to the uniquely distinct features. The next set of result is irrelevant.

Search Engine Performance and its Improvement

The results obtained are quite quick and there is no heavy process overload. It is a memory efficient code well organised in python using libraries of OpenCV and Numpy.

Some of the improvements that could be looked upon are :

- ➔ **Not Scalable :This current approach isn't that scalable. In order to make it scale to 100 thousand images one would either have to put this behind a MapReduce cluster or generate clusters of similar color distributions and then only search within the relevant clusters.**
- ➔ **Susceptible to Outliers: If two images are taken from a distance then the image has a large amount of grass and sky areas around it as a result building matching fails and then two images land up to having the same similarity metric even if they are of different buildings. RANSAC can be used to remove such outliers.**

Quality of the code and its Organization

The code is well documented providing comments for almost every line of the code. This helps in understanding the process of how images are indexed and then later on when a query image is fed , how a matching list of images are obtained.

The code is very well structured developed using object oriented concepts. This is one of the major reason of quick response time of this image search engine.

The 4 steps to build a Image based Search Engine is well defined :

1. **Defining a descriptor : 3D color histogram in the RGB color space with 8 bins per red, green, and blue channel.**
2. **Indexing the dataset : The 3D RGB histogram will be applied to each image in the dataset. This simply means , the code loops over 25 image dataset, extract a 3D RGB histogram from each image, store the features in a dictionary, and write the dictionary to file.**
3. **Defining similarity metric: Two feature vectors can be compared using a distance metric. A chi-squared distance metric is used. Images will be considered identical if**

their feature vectors have a chi-squared distance of zero. The larger the distance gets, the less similar they are.

4. **Searching: To perform a search, the .cpickle file generated after indexing is used to retrieve the results. The distance metric is used to rank how similar images are in the index list to query images. Results are then sorted via similarity**

Finally, path of the top 10 images are fetched and they are displayed to user in two montages, top 5 in 1st and 6-10 in 2nd.

Take Away

- ➔ **One of the major takeaway from this experiment is that redundant pixels like sky, greenery, road, same color of building, glass windows, etc. could sometimes create an illusion for the code to detect two images as similar even if they are of different places: Like we saw for the case of The Commons Building and Capen Hall.**
- ➔ **The resulting matrix size was fixed at 400 x 166 as a result the all the images had to be reduced to that size leading deterioration of the image quality and loss in formation.**
- ➔ **Since the photographs were taken at different time during the day and at different angles the Intensity and Exposure to Light is different sometimes affecting the similarity metric.**
- ➔ **This code helps us better understand the process of Image Retrieval making it simple to execute with different real world images.**

Conclusion

In this project we've explored how to create an image search engine from start to finish. The first step was to choose an image descriptor — we used a 3D RGB histogram to characterize the color of our images. We then indexed each image in our dataset using our descriptor by extracting feature vectors (i.e. the histograms). From there, we used the chi-squared distance to define "similarity" between two images. Finally, we glued all the pieces together and created a University of Buffalo image search engine for 5 landmarks.