Project 2: Content-based Image Retrieval

Due Feb 10 by 11:59pm **Points** 30

The purpose of this project is to continue the process of learning how to manipulate and analyze images at a pixel level. In addition, this is the first project where we will be doing matching, or pattern recognition.

The overall task for this project is, given a database of images and a target image, find images in the data with similar content. For this project we using both classic features and deep network embeddings. The classic features will be generic characteristics of the images such as color, texture, and their spatial layout. The deep network embedding will be features extracted from a ResNet18 network. This will give you practice with working with different color spaces, histograms, spatial features, texture features, and embeddings.

A useful reference for this project is **Chapter 8** **(https://northeastern.instructure.com/courses/175350/files/24760257?wrap=1)** of Computer Vision by Shapiro and Stockman. You can find **pdfs** **(https://courses.cs.washington.edu/courses/cse576/book/)** for each chapter here.

Setup

You can find a large database of images **here** **(https://northeastern.instructure.com/courses/175350/files/24760251?wrap=1)** **(https://northeastern.instructure.com/courses/175350/files/24760251/download?download_frd=1)** . The images are a variety of sizes, ages, and quality. You may want to set aside a subset of the images (e.g. 10-20 of them) for basic testing as you develop your code. Pick a set that have some similar images and some very different images.

Here is **code for reading all of the image files in a directory** **(https://northeastern.instructure.com/courses/175350/files/25926077?wrap=1)** **(https://northeastern.instructure.com/courses/175350/files/25926077/download?download_frd=1)** . You can modify it or pull from it as you see fit. This project should continue to use C++, and you may find some of your code from the prior assignment to be useful.

Tasks

For this task the inputs to the system will be a target image T, an image database B, a method of computing features for an image F, a distance metric for comparing the image features from two images

D(Ft,Fi), and the desired number of output images N. The output will be an ordered list of the N images in B that are most similar to T, according to F and D, in ascending order of distance. Remember, the smaller a distance, the more similar two images will be.

The process can be generally described as the following four steps.

1. Compute the features Ft on the target image T.
2. Compute the features {Fi} on all of the images in B.
3. Compute the distance of T from all of the images in B using the distance metric D(Ft,Fi).
4. Sort the images in B according to their distance from T and return the best N matches.

If you are executing many queries with different target images, step 2 can be computed off-line once, stored in a database, and then used for many different target images. You can store the features in a binary file or in something like a CSV file (comma-separated-values), which is a simple text file. Here is some **code (https://northeastern.instructure.com/courses/175350/files/24760260?wrap=1) (https://northeastern.instructure.com/courses/175350/files/24760260/download?download_frd=1)** for writing and reading CSV files in a particular format that may be helpful.

The entire process can be implemented as a command line program that takes in a target filename for T, a directory of images as the database B, the feature type, the matching method, and the number of images N to return. The program should print out the filenames of the top N matching images. If you're feeling fancy, you could create an OpenCV GUI that lets the user open an image and then displays the target and the top N matches. Hardcoding files or directories in your code will ultimately make it less efficient to run multiple experiments.

1. **Baseline Matching:**

   Use the 7x7 square in the middle of the image as a feature vector. Use sum-of-squared-difference as the distance metric. Make sure that comparing the image with itself results in a distance of 0.

   Use this task to get your overall pipeline working. As noted above you can proceed in one of two ways. The first method is to make a single program that does everything. The second method is to split the process into two pieces. The first method is simpler, but it will run much more slowly each time you want to make a query. The two methods are described in more detail below.

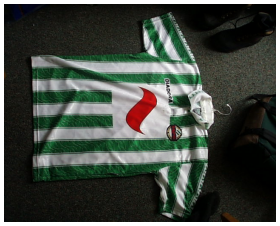
   1. Make a single program that (1) reads the target image and computes its features, (2) loops over the directory of images, and for each image computes the features and compares them to the target image, storing the result in an array or vector, and (3) sorts the list of matches and returns the top N. You could write a separate program for each feature set or one program that takes which feature set to use as an argument.
   2. Make two programs. The first program is given a directory of images and feature set and it writes the feature vector for each image to a file (**example code (https://northeastern.instructure.com/courses/175350/files/24760260?wrap=1) (https://northeastern.instructure.com/courses/175350/files/24760260/download?download_frd=1)**).

The second program is given a target image, the feature set, and the feature vector file. It then computes the features for the target image, reads the feature vector file, and identifies the top N matches.

Required result 1: show the top three matches for the target image pic.1016.jpg. My top three matches are pic.0986.jpg, pic.0641.jpg, and pic.0233.jpg.


pic.1016.jpg


pic.0986.jpg


pic.0641.jpg

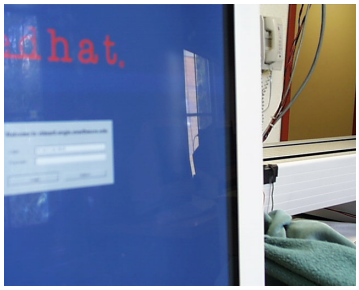

pic.0547.jpg

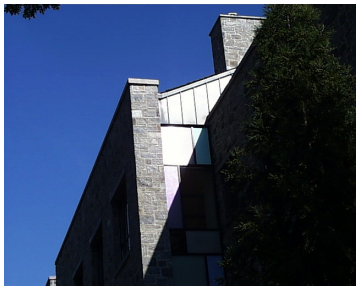
2. **Histogram Matching:**

Use a single normalized color histogram of your choice (has to be at least two-dimensional) as the feature vector. Use histogram intersection as the distance metric. Write your own code to calculate the histograms from the image. You can use the cv::Mat to hold the 2-D or 3-D histogram data. Write your own distance metric code.

Required results 2: show the top three matches for the target image pic.0164.jpg.


pic.0164.jpg

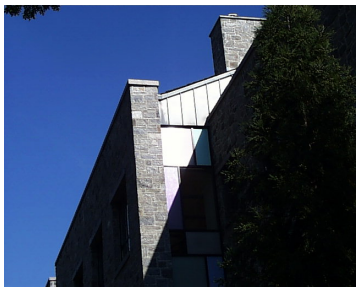

pic.0080.jpg


pic.1032.jpg


pic.0461.jpg

These results for query pic.0164 were obtained with a whole image rg chromaticity histogram using 16 bins for each of r and g and histogram intersection as the distance metric. These are intended as sample results only in case you want to check your code's functionality. You can choose what type of histogram you want to use, and different histograms and distance metrics will produce different results.






pic.0164.jpg pic.0110.jpg pic.1032.jpg pic.0092.jpg

These results for query pic.0164 were obtained with a whole image RGB histogram using 8 bins for each of RGB and histogram intersection as the distance metric.

3. **Multi-histogram Matching:**

   Use two or more color histograms of your choice as the feature vector. The histograms should represent different spatial parts of the image. The parts can be overlapping or disjoint. For example, you could use a whole-image histogram and a second histogram that looks at only the center of the image. Use a distance metric of your own design. For example, you could use histogram intersection to compare the corresponding histograms and then use weighted averaging to combine the distances between the different histograms.

   Required results 3: show the top three matches for the target image pic.0274.jpg.




pic.0274.jpg pic.0273.jpg pic.1031.jpg pic.0409.jpg

These results for query pic.0274 were obtained with two RGB histograms, representing the top and bottom halves of the image, using 8 bins for each of RGB and histogram intersection as the distance metric.

4. **Texture and Color:**

   Use a whole image color histogram and a whole image texture histogram as the feature vector. Choose a texture metric of your choice for this task. Design a distance metric that weights the two types of histograms equally.

   The simplest texture metric to implement is to calculate the Sobel magnitude image and use a histogram of gradient magnitudes as your texture feature. I strongly recommend doing this first, since

you already have the Sobel filters from project 1. A slightly more sophisticated method is to add a histogram of gradient orientations, or to make a 2D histogram of gradient orientation and magnitude.

Good extensions would be to implement one or more additional texture analysis methods and test their effects (alone, or in combination with color information). You may use openCV functions wherever possible for this task. Some other texture feature methods that are filter or histogram-based include:

- Features of one or more co-occurrence matrices: energy, entropy, contract, homogeneity, and max probability.
- Histograms of Laws filter responses.
- Histograms of Gabor filter responses.
- Features extracted from a Fourier Transform of the image, such as resizing the power spectrum to a 16x16 image.

Required results 4: show the top three matches for the target image pic.0535.jpg and show how they differ when compared to tasks 2 and 3.

Definitely not required, but if you want to explore further: **Varma and Zisserman, CVPR 2003 (https://northeastern.instructure.com/courses/175350/files/24760292?wrap=1)** .

5. **Deep Network Embeddings**

   Use the feature vectors contained in this **csv file (https://northeastern.instructure.com/courses/175350/files/25926071?wrap=1)** ↓ **(https://northeastern.instructure.com/courses/175350/files/25926071/download?download_frd=1)** . Each row of the file contains the filename in column 1, followed by 512 values. The 512 values are the output of the final average pooling layer of a ResNet18 deep network pre-trained on ImageNet. ImageNet is a 1M image database with 1k categories of diverse types.

   Select a distance metric to use for matching the embedding vectors for two images. One option is to use sum-square distance. Another option is to use cosine-distance, which sometimes works better for high dimensional spaces. Cosine distance is d(v1, v2) = 1 - cos(theta) where theta is the angle between the two vectors. You can calculate the cosine of the angle between two vectors by first normalizing each vector by its length (Euclidean distance or L2-norm) and then taking the dot product of the normalized vectors.

   Note, unlike the other features, you will need to get the feature vector for the target image from the file, rather than calculate it dynamically.

   Required results 5: include the top 3 results for images pic.0893.jpg and pic.0164.jpg and compare the results with the prior methods.

6. **Compare DNN Embeddings and Classic Features**

Pick 2-3 images and compare the results of using DNN Embeddings versus classic features. Some interesting images to try are 1072, 948, and 734. Is the DNN embedding vector always better? This is a subjective question, so think about your answer and support it with examples.

Required result 6: compare and contrast the DNN embedding and classic features results for 2-3 images of your choice.

7. **Custom Design:**

   Choose a specific type of image for which you want to execute CBIR. For example, you could choose sunsets, or you could choose pictures that contain bananas. Design a feature vector and distance metric of your choice. You may include the DNN feature vector in your method, but it cannot be the only feature used by your method.

   During development, it can be handy to have a small training set so you don't have to run your experiments on the whole database every time you make a change. Having some similar images and some dissimilar images in your training set will help you to design a metric that orders them reasonably.

   To evaluate your method, pick a couple of queries from the remaining data and decide what you think the answer should be for N images. Evaluate your algorithm on these examples and summarize your results.

   Required results 7: for two target images of your choice, show the top five results. It's also helpful to show some of the least similar results. If you want to show a video of your system in action, include a link to the video in your readme file.

Extensions

- Create additional features and matching methods. Compare and contrast your different methodologies.
- Design a GUI for your system or otherwise make it more sophisticated than a command line program.
- See if you can find as many pictures with bananas as possible given an input image containing a banana. The spatial variance of different colors can be a useful metric when looking for images with similar size blobs of similar colors.
- There are also lots of pictures of blue trash can bins. How many of these can your system recall given a target image that contains one?
- You have a face detector, what happens if you have a metric that uses faces if there is a face in the target image?
- Be creative.

Report

When you are done with your project, write a short report that demonstrates the functionality of each task. You can write your report in the application of your choice, but you need to submit it **as a pdf** along with your code. Your report should have the following structure. Please **do not include code** in your report.

1. A short description of the overall project in your own words. (200 words or less)
2. Any required images along with a short description of the meaning of the image.
3. A description and example images of any extensions.
4. A short reflection of what you learned.
5. Acknowledgement of any materials or people you consulted for the assignment.

Submission

Submit your code and report to **Gradescope** (https://www.gradescope.com) . When you are ready to submit, upload your code, report, and a readme file. The readme file should contain the following information.

- Your name and any other group members, if any.
- Links/URLs to any videos you created and want to submit as part of your report.
- What operating system and IDE you used to run and compile your code.
- Instructions for running your executables.
- Instructions for testing any extensions you completed.
- Whether you are using any time travel days and how many.

For project 2, submit your .cpp and .h (.hpp) files, pdf report, and readme.txt (readme.md). Note, if you find any errors or need to update your code, you can resubmit as many times as you wish up until the deadline.

As noted in the syllabus, projects submitted by the deadline can receive full credit for the base project and extensions. (max 30/30). Projects submitted up to a week after the deadline can receive full credit for the base project, but not extensions (max 26/30). You also have eight time travel days you can use during the semester to adjust any deadline, using up to three days on any one assignment (no fractional days). If you want to use your time travel days, email the instructor prior to the deadline for which you plan to use them. If you need to make use of the "stuff happens" clause of the syllabus, contact the instructor as soon as possible to make alternative arrangements.

Receiving grades and feedback

After your project has been graded, you can find your grade and feedback on Gradescope. Pay attention to the feedback, because it will probably help you do better on your next assignment.