Course Project Tutorial 4

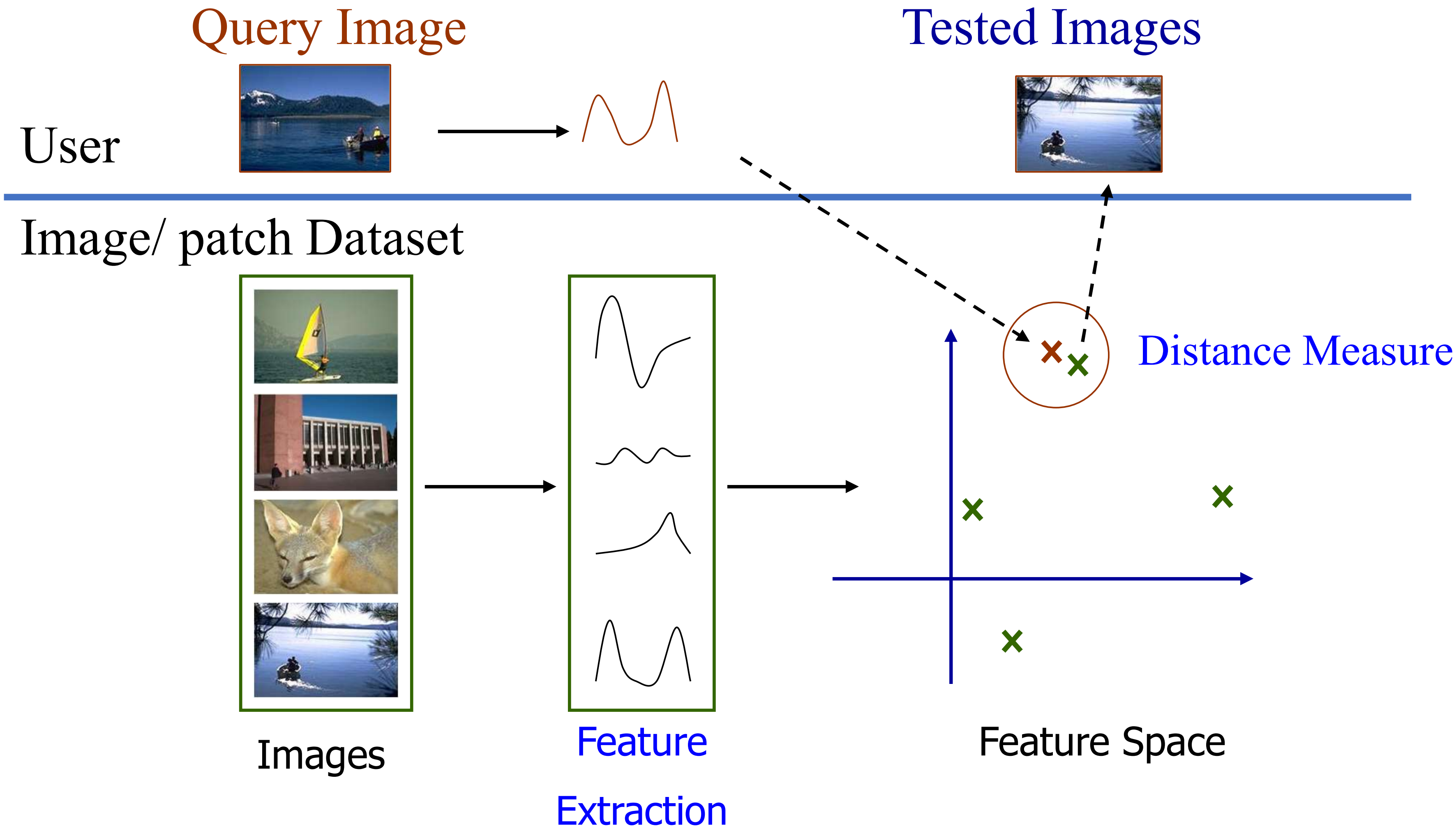
(Advanced Requirements)

CS4185 Multimedia Technologies and Applications

Advanced Requirements:
Program -> application

- Technical improvements
  - New retrieval algorithms
  - Feature restoration
  - Use crawler to expand the dataset
  - ……
- UI design
  - MFC UI
  - Web UI
  - ……

Image Features / Distance Measures



Query Image

Tested Images

User

Image/ patch Dataset

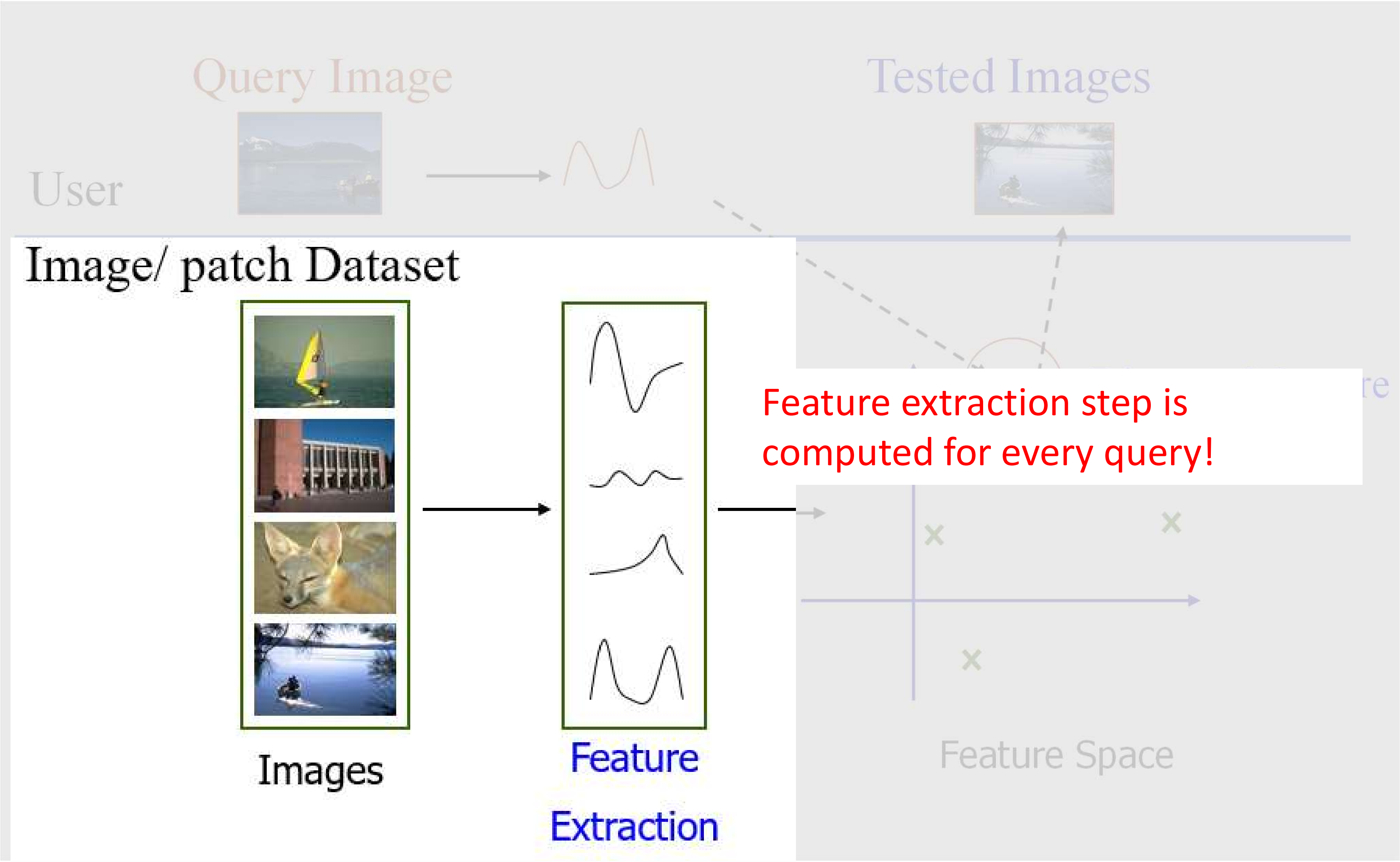
Distance Measure

Images

Feature Extraction

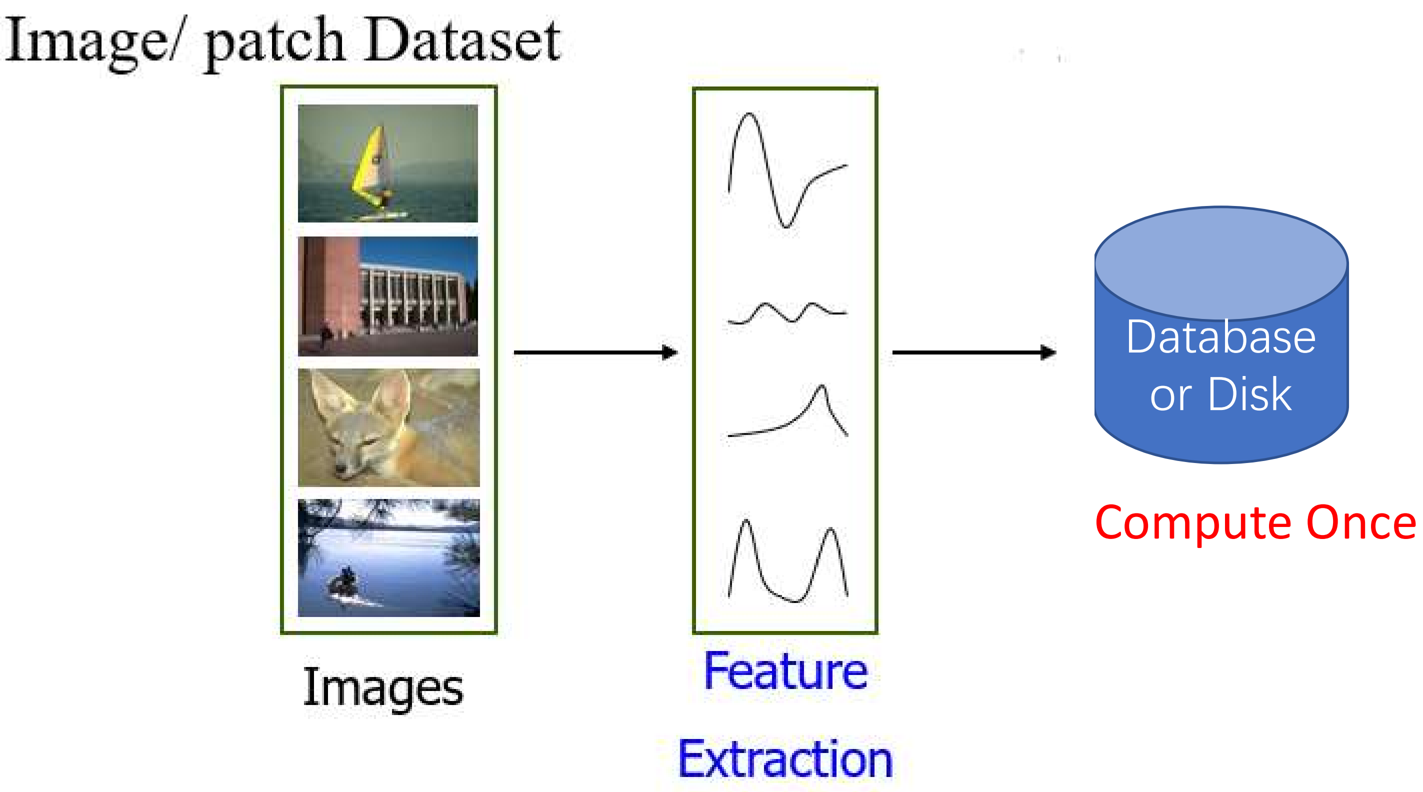
Feature Space

Feature Restoration to Speed Up Your Program



Feature extraction step is computed for every query!

Feature extraction phase



Image/ patch Dataset

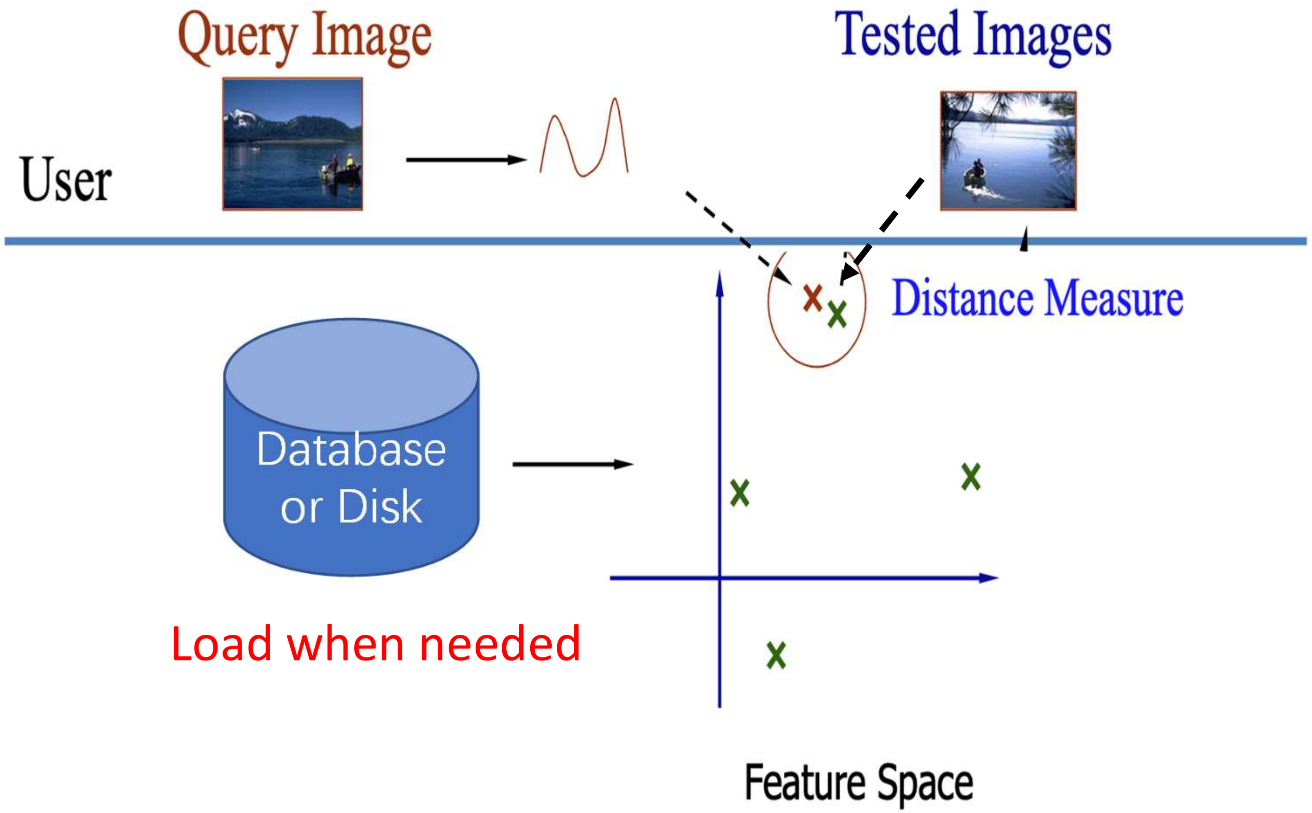
Images

Feature Extraction

Database or Disk

Compute Once

Inference phase



Load when needed

Advantages

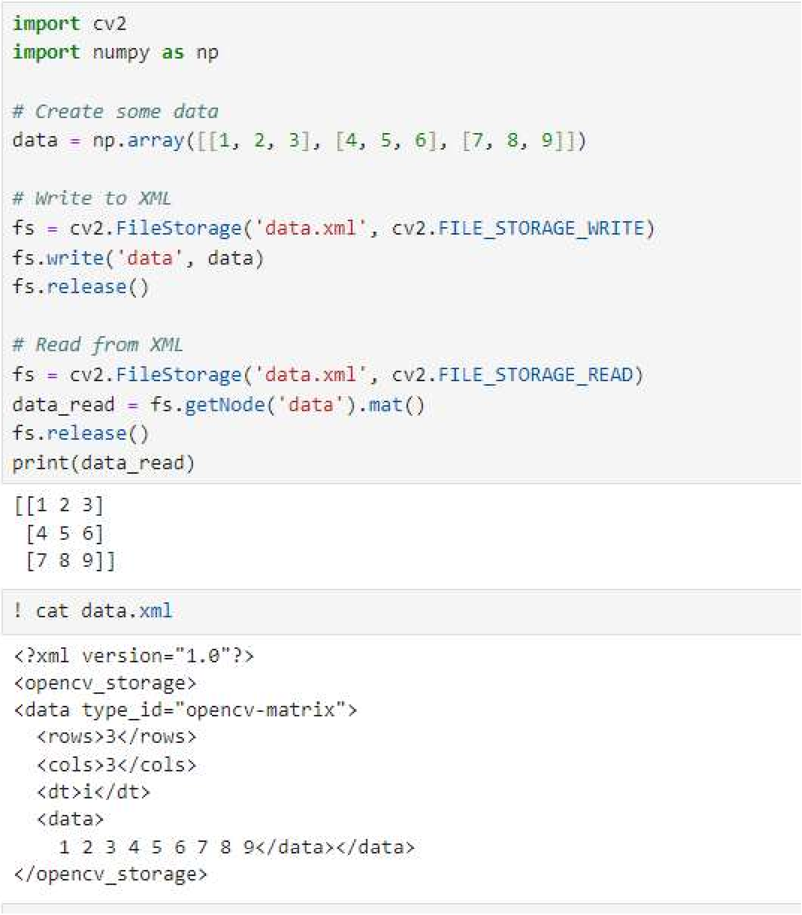
- No need to extract features of all images in inference time
- Speed up the program
- Program -> application

One Solution

OpenCV in python supports storing data using several formats, such as XML and YAML. You can also use the python standard module, such as Pickle and JSON to store data:

1. XML/YAML: OpenCV provides the cv::FileStorage class, which allows you to read and write data in XML or YAML format. This is particularly useful for storing complex data structures like matrices and custom objects.
2. Pickle: Python’s pickle module is a standard way to serialize and deserialize Python objects. It’s useful for saving the state of objects, such as machine learning models, to disk for later use.
3. JSON: The json module in Python is another option for serialization. JSON is a lightweight, human-readable format that’s widely used for data exchange.

OpenCV-FileStorage



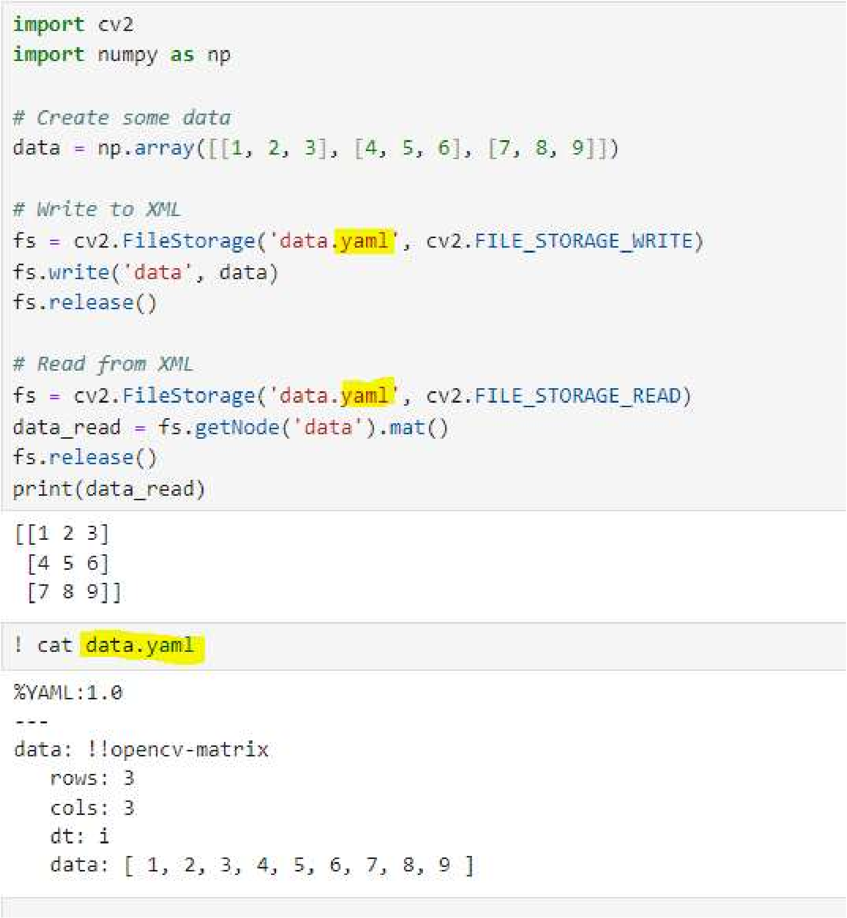
```python
import cv2
import numpy as np

# Create some data
data = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Write to XML
fs = cv2.FileStorage('data.xml', cv2.FILE_STORAGE_WRITE)
fs.write('data', data)
fs.release()

# Read from XML
fs = cv2.FileStorage('data.xml', cv2.FILE_STORAGE_READ)
data_read = fs.getNode('data').mat()
fs.release()
print(data_read)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
! cat data.xml
```

```
<?xml version="1.0"?>
<opencv_storage>
<data type_id="opencv-matrix">
  <rows>3</rows>
  <cols>3</cols>
  <dt>i</dt>
  <data>
    1 2 3 4 5 6 7 8 9</data></data>
</opencv_storage>
```

XML format



```python
import cv2
import numpy as np

# Create some data
data = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Write to XML
fs = cv2.FileStorage('data.yaml', cv2.FILE_STORAGE_WRITE)
fs.write('data', data)
fs.release()

# Read from XML
fs = cv2.FileStorage('data.yaml', cv2.FILE_STORAGE_READ)
data_read = fs.getNode('data').mat()
fs.release()
print(data_read)
```

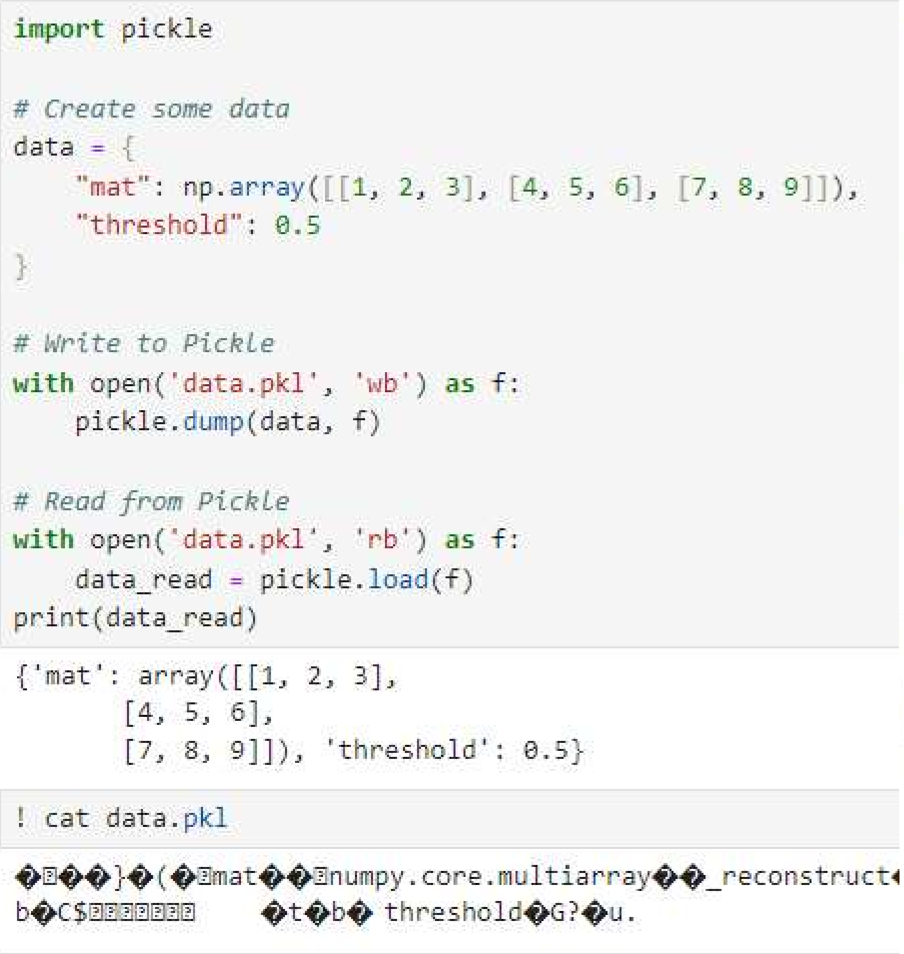
```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
! cat data.yaml
```

```
%YAML:1.0
---
data: !!opencv-matrix
   rows: 3
   cols: 3
   dt: i
   data: [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ]
```

YAML format

Pickle & Json



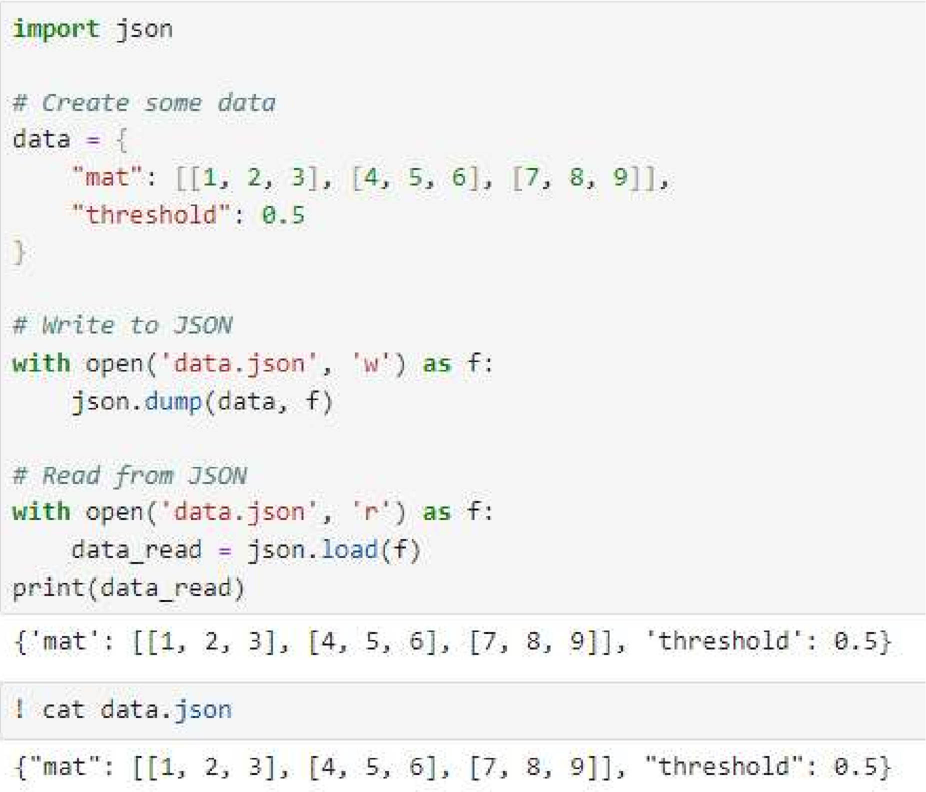
```python
import pickle

# Create some data
data = {
    "mat": np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]),
    "threshold": 0.5
}

# Write to Pickle
with open('data.pkl', 'wb') as f:
    pickle.dump(data, f)

# Read from Pickle
with open('data.pkl', 'rb') as f:
    data_read = pickle.load(f)
print(data_read)
```

```
{'mat': array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]]), 'threshold': 0.5}
```

```
! cat data.pkl
```

```
�▯��}�(�▯mat��▯numpy.core.multiarray��_reconstruct�
b�C$▯▯▯▯▯▯▯    �t�b� threshold�G?�u.
```

Pickle module can store any python objects in binary format.



```python
import json

# Create some data
data = {
    "mat": [[1, 2, 3], [4, 5, 6], [7, 8, 9]],
    "threshold": 0.5
}

# Write to JSON
with open('data.json', 'w') as f:
    json.dump(data, f)

# Read from JSON
with open('data.json', 'r') as f:
    data_read = json.load(f)
print(data_read)
```

```
{'mat': [[1, 2, 3], [4, 5, 6], [7, 8, 9]], 'threshold': 0.5}
```

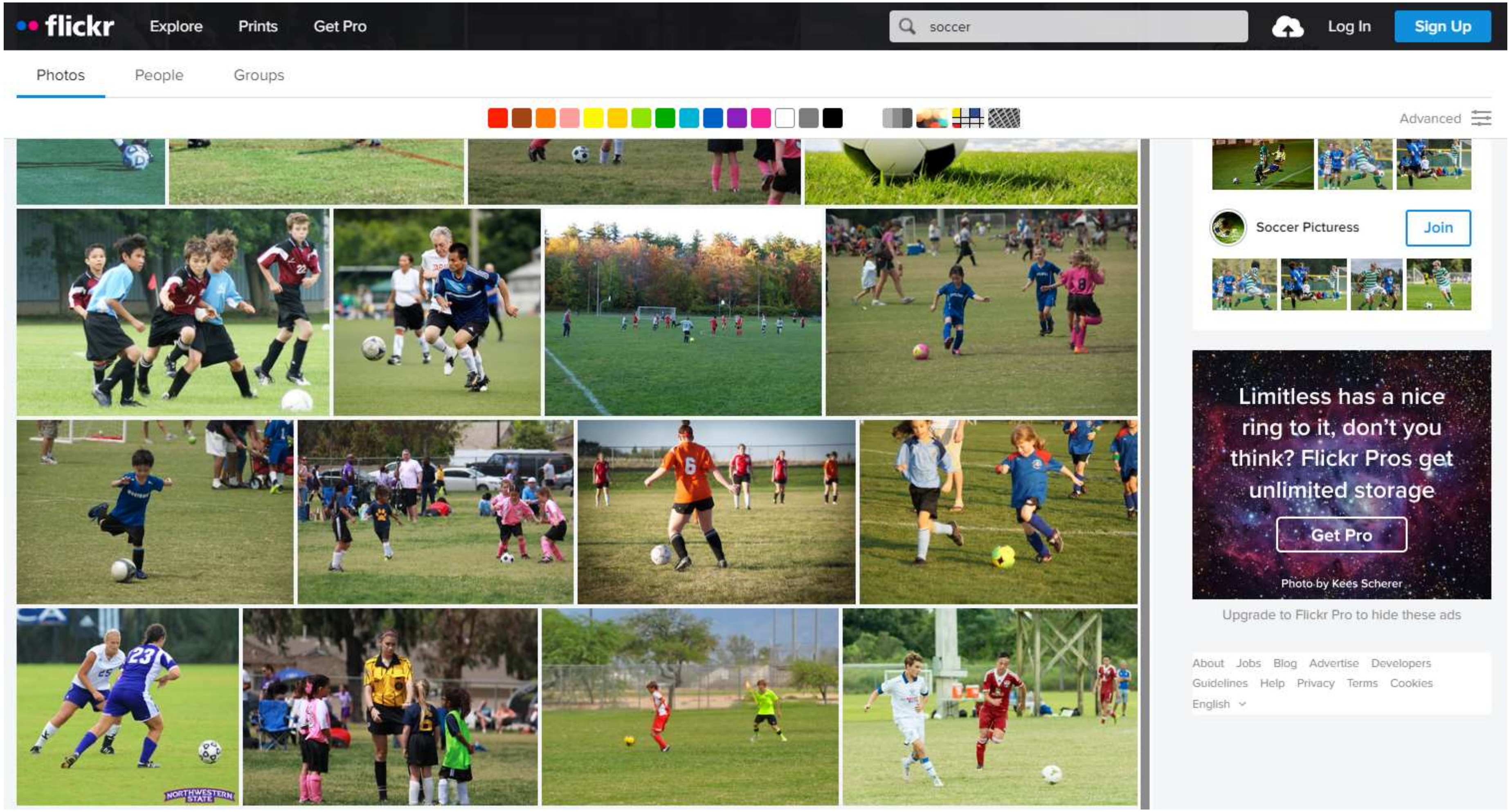
```
! cat data.json
```

```
{"mat": [[1, 2, 3], [4, 5, 6], [7, 8, 9]], "threshold": 0.5}
```

Json module can store common python objects in human-readable format.

Use a web crawler to extend the dataset

crawl anything you want

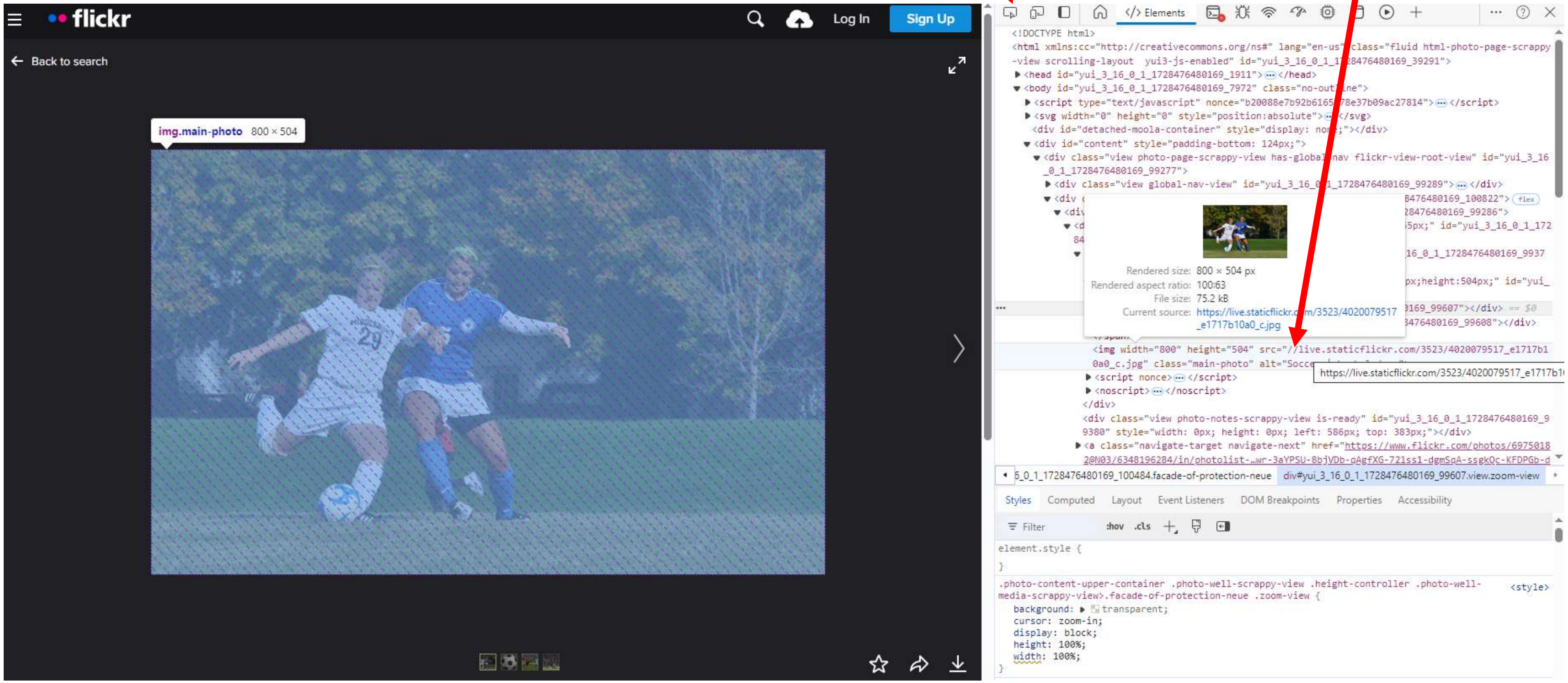


Limitless has a nice ring to it, don't you think? Flickr Pros get unlimited storage

Step2: Use chrome dev-tool to locate the address of an image

1. Click on this button to select the image

2. Locate the image URL



```
<!DOCTYPE html>
<html xmlns:cc="http://creativecommons.org/ns#" lang="en-us" class="fluid html-photo-page-scrappy
-view scrolling-layout  yui3-js-enabled" id="yui_3_16_0_1_1728476480169_39291">
 ▶<head id="yui_3_16_0_1_1728476480169_1911">⋯</head>
 ▼<body id="yui_3_16_0_1_1728476480169_7972" class="no-outline">
   ▶<script type="text/javascript" nonce="b20088e7b92b6165 78e37b09ac27814">⋯</script>
   ▶<svg width="0" height="0" style="position:absolute">⋯</svg>
    <div id="detached-moola-container" style="display: none;"></div>
   ▼<div id="content" style="padding-bottom: 124px;">
     ▼<div class="view photo-page-scrappy-view has-global nav flickr-view-root-view" id="yui_3_16
      _0_1_1728476480169_99277">
       ▶<div class="view global-nav-view" id="yui_3_16_0_1_1728476480169_99289">⋯</div>
       ▼<div                                                8476480169_100822"> flex
         ▼<div                                              28476480169_99286">
           ▼<d                                              5px;" id="yui_3_16_0_1_172
            84
             ▼                                              16_0_1_1728476480169_9937

                                                            px;height:504px;" id="yui_
                                                            169_99607"></div> == $0
                                                            3476480169_99608"></div>
              </span>
              <img width="800" height="504" src="//live.staticflickr.com/3523/4020079517_e1717b1
              0a0_c.jpg" class="main-photo" alt="Socce
             ▶<script nonce>⋯</script>
             ▶<noscript>⋯</noscript>
            </div>
            <div class="view photo-notes-scrappy-view is-ready" id="yui_3_16_0_1_1728476480169_9
            9380" style="width: 0px; height: 0px; left: 586px; top: 383px;"></div>
           ▶<a class="navigate-target navigate-next" href="https://www.flickr.com/photos/6975018
            2@N03/6348196284/in/photolist-…wr-3aYPSU-8bjVDb-qAgfXG-721ss1-dgmSqA-ssgkQc-KFDPGb-d
```

Rendered size: 800 × 504 px
Rendered aspect ratio: 100:63
File size: 75.2 kB
Current source: https://live.staticflickr.com/3523/4020079517_e1717b10a0_c.jpg

https://live.staticflickr.com/3523/4020079517_e1717b1

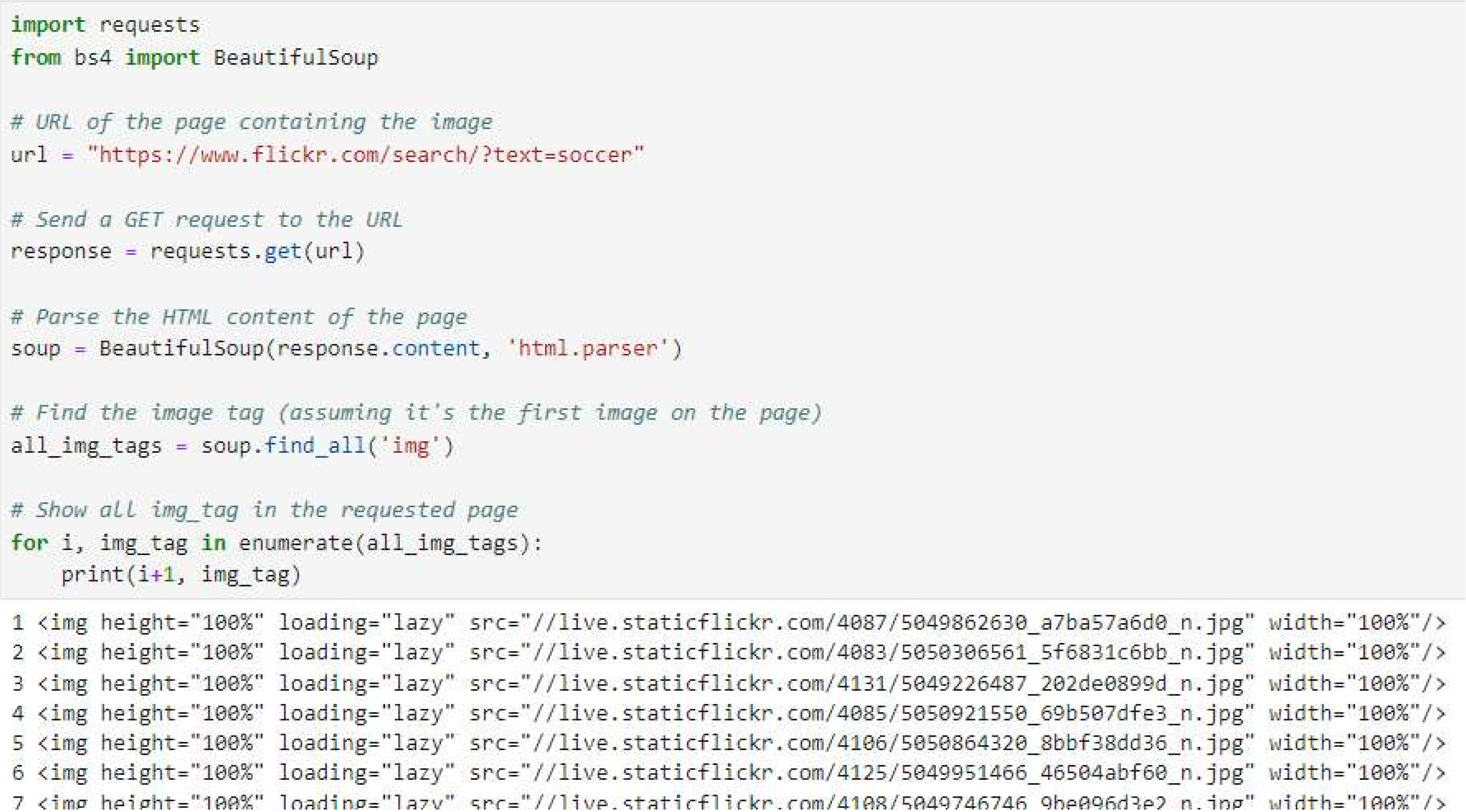
6_0_1_1728476480169_100484.facade-of-protection-neue   div#yui_3_16_0_1_1728476480169_99607.view.zoom-view

```
element.style {
}
.photo-content-upper-container .photo-well-scrappy-view .height-controller .photo-well-      <style>
media-scrappy-view>.facade-of-protection-neue .zoom-view {
  background: ▶ transparent;
  cursor: zoom-in;
  display: block;
  height: 100%;
  width: 100%;
}
```

Step3: Send http request to get images in an automatic manner

Possible Solutions:

1. `Requests` **(**`pip install requests`**) +** `BeautifulSoup` **(**`pip install beautifulsoup4`**)**
2. **Some python-crawler package, likes Scrapy, Crawlee and Image-Crawler.**



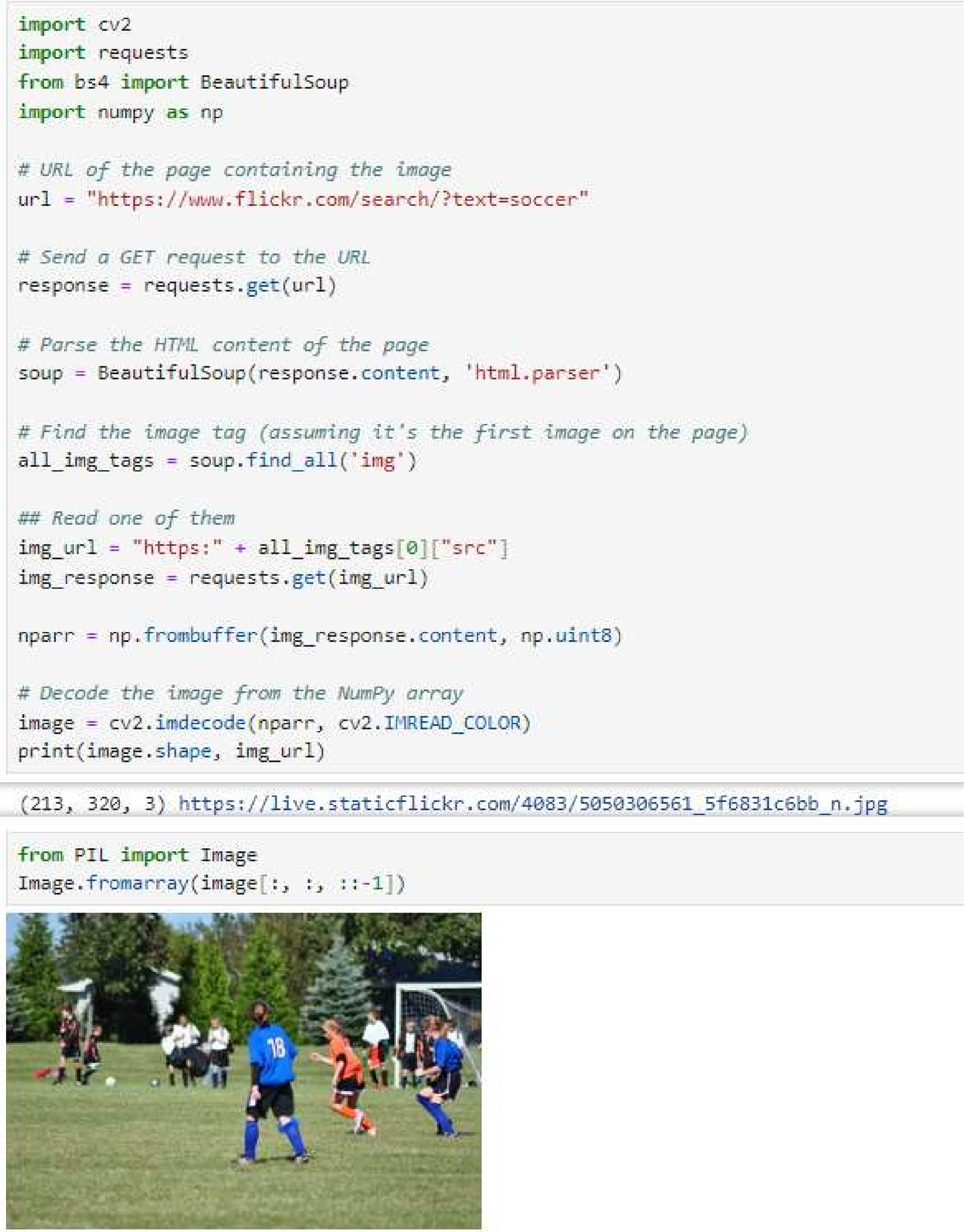
```python
import requests
from bs4 import BeautifulSoup

# URL of the page containing the image
url = "https://www.flickr.com/search/?text=soccer"

# Send a GET request to the URL
response = requests.get(url)

# Parse the HTML content of the page
soup = BeautifulSoup(response.content, 'html.parser')

# Find the image tag (assuming it's the first image on the page)
all_img_tags = soup.find_all('img')

# Show all img_tag in the requested page
for i, img_tag in enumerate(all_img_tags):
    print(i+1, img_tag)
```

```
1 <img height="100%" loading="lazy" src="//live.staticflickr.com/4087/5049862630_a7ba57a6d0_n.jpg" width="100%"/>
2 <img height="100%" loading="lazy" src="//live.staticflickr.com/4083/5050306561_5f6831c6bb_n.jpg" width="100%"/>
3 <img height="100%" loading="lazy" src="//live.staticflickr.com/4131/5049226487_202de0899d_n.jpg" width="100%"/>
4 <img height="100%" loading="lazy" src="//live.staticflickr.com/4085/5050921550_69b507dfe3_n.jpg" width="100%"/>
5 <img height="100%" loading="lazy" src="//live.staticflickr.com/4106/5050864320_8bbf38dd36_n.jpg" width="100%"/>
6 <img height="100%" loading="lazy" src="//live.staticflickr.com/4125/5049951466_46504abf60_n.jpg" width="100%"/>
7 <img height="100%" loading="lazy" src="//live.staticflickr.com/4108/5049746746_9be096d3e2_n.jpg" width="100%"/>
```

Crawl Data

A full example using:

- Requests
- BeautifulSoup

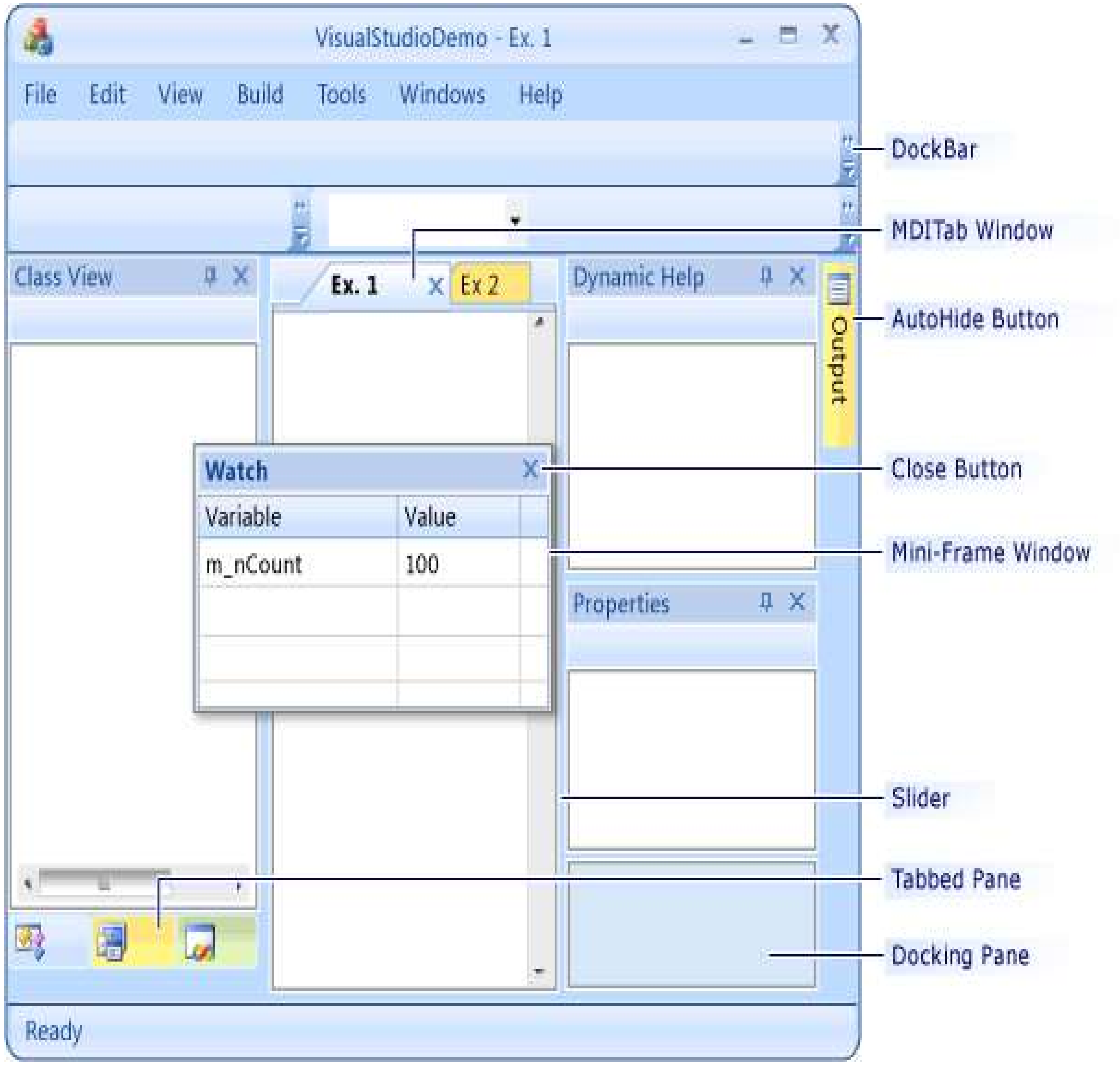


```
import cv2
import requests
from bs4 import BeautifulSoup
import numpy as np

# URL of the page containing the image
url = "https://www.flickr.com/search/?text=soccer"

# Send a GET request to the URL
response = requests.get(url)

# Parse the HTML content of the page
soup = BeautifulSoup(response.content, 'html.parser')

# Find the image tag (assuming it's the first image on the page)
all_img_tags = soup.find_all('img')

## Read one of them
img_url = "https:" + all_img_tags[0]["src"]
img_response = requests.get(img_url)

nparr = np.frombuffer(img_response.content, np.uint8)

# Decode the image from the NumPy array
image = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
print(image.shape, img_url)
```

```
(213, 320, 3) https://live.staticflickr.com/4083/5050306561_5f6831c6bb_n.jpg
```

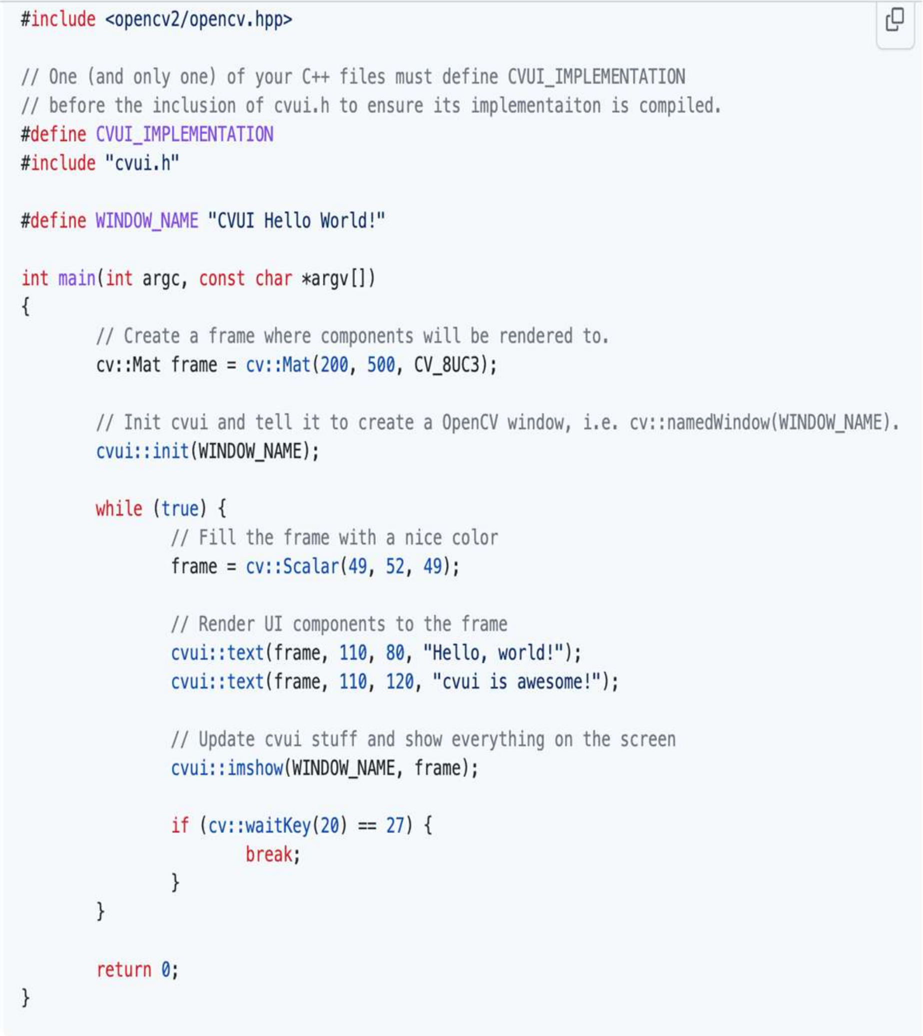
```
from PIL import Image
Image.fromarray(image[:, :, ::-1])
```

MFC UI

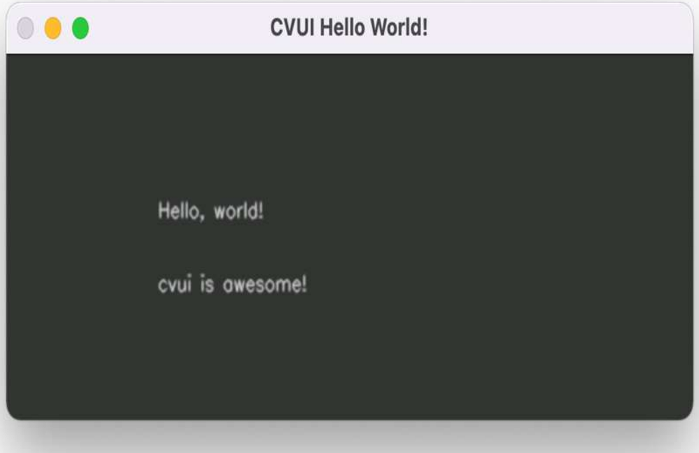


VisualStudioDemo - Ex. 1
File
Edit
View
Build
Tools
Windows
Help
Class View
Ex. 1
Ex 2
Dynamic Help
Output
Watch
Variable
Value
m_nCount
100
Properties
Ready
DockBar
MDITab Window
AutoHide Button
Close Button
Mini-Frame Window
Slider
Tabbed Pane
Docking Pane

UI Design with cvui



```
#include <opencv2/opencv.hpp>

// One (and only one) of your C++ files must define CVUI_IMPLEMENTATION
// before the inclusion of cvui.h to ensure its implementaiton is compiled.
#define CVUI_IMPLEMENTATION
#include "cvui.h"

#define WINDOW_NAME "CVUI Hello World!"

int main(int argc, const char *argv[])
{
        // Create a frame where components will be rendered to.
        cv::Mat frame = cv::Mat(200, 500, CV_8UC3);

        // Init cvui and tell it to create a OpenCV window, i.e. cv::namedWindow(WINDOW_NAME).
        cvui::init(WINDOW_NAME);

        while (true) {
                // Fill the frame with a nice color
                frame = cv::Scalar(49, 52, 49);

                // Render UI components to the frame
                cvui::text(frame, 110, 80, "Hello, world!");
                cvui::text(frame, 110, 120, "cvui is awesome!");

                // Update cvui stuff and show everything on the screen
                cvui::imshow(WINDOW_NAME, frame);

                if (cv::waitKey(20) == 27) {
                        break;
                }
        }

        return 0;
}
```



```
Hello, world!

cvui is awesome!
```

Here, frame containing Components is a cv::Mat

Image

`cvui::image()` renders an image, i.e. `cv::Mat`. The signature of the function is:

```
void image(cv::Mat& theWhere, int theX, int theY, cv::Mat& theImage);
```

where `theWhere` is the image/frame where the image will be rendered, `theX` is the position X, `theY` is the position Y, and `theImage` is an image to be rendered in the specified destination.

Below is an example showing an image being loaded then displayed using `cvui::image()`. The result on the screen is shown in Figure 1.

```
cv::Mat lena_face = cv::imread("lena_face.jpg", cv::IMREAD_COLOR);
cvui::image(frame, 10, 10, lena_face);
```

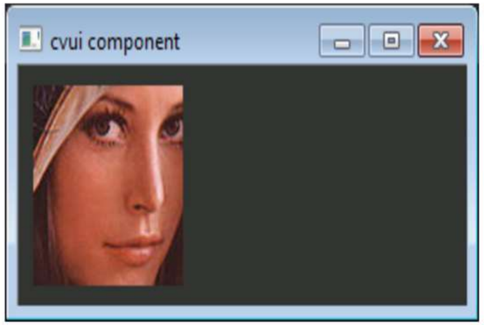
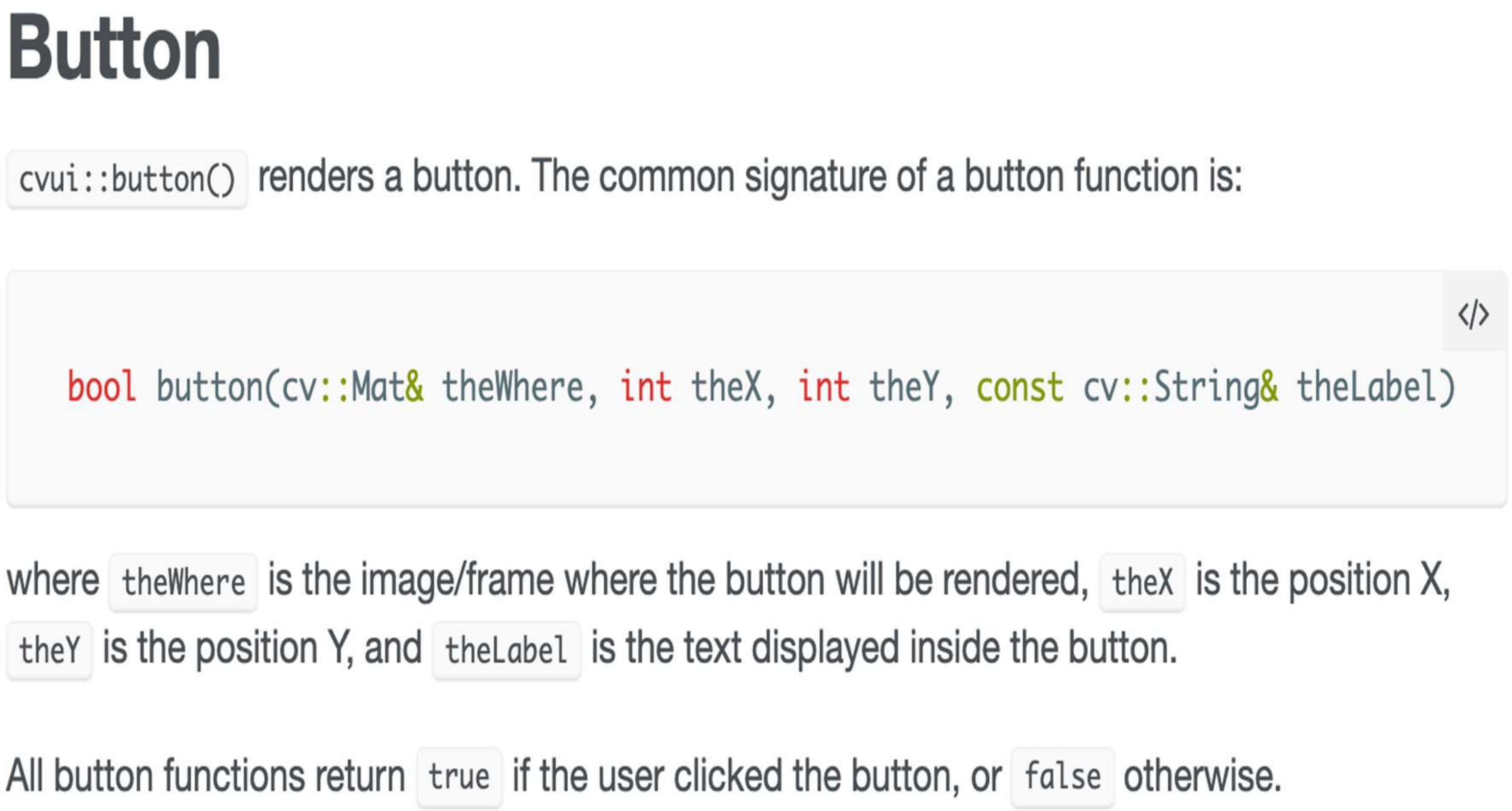


Figure 1: image `lena_face.jpg` displayed on the screen.



Button

`cvui::button()` renders a button. The common signature of a button function is:

```
bool button(cv::Mat& theWhere, int theX, int theY, const cv::String& theLabel)
```

where `theWhere` is the image/frame where the button will be rendered, `theX` is the position X, `theY` is the position Y, and `theLabel` is the text displayed inside the button.

All button functions return `true` if the user clicked the button, or `false` otherwise.

Button width will auto-adjust based on the size of its label. Below is an example of a button with auto-adjusted width (shown in Figure 1):

```
// cv::Mat frame, x, y, label
if (cvui::button(frame, 100, 40, "Button")) {
    // button was clicked
}
```

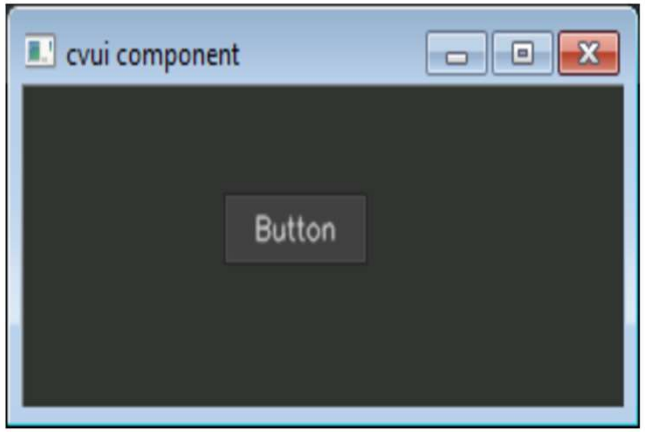
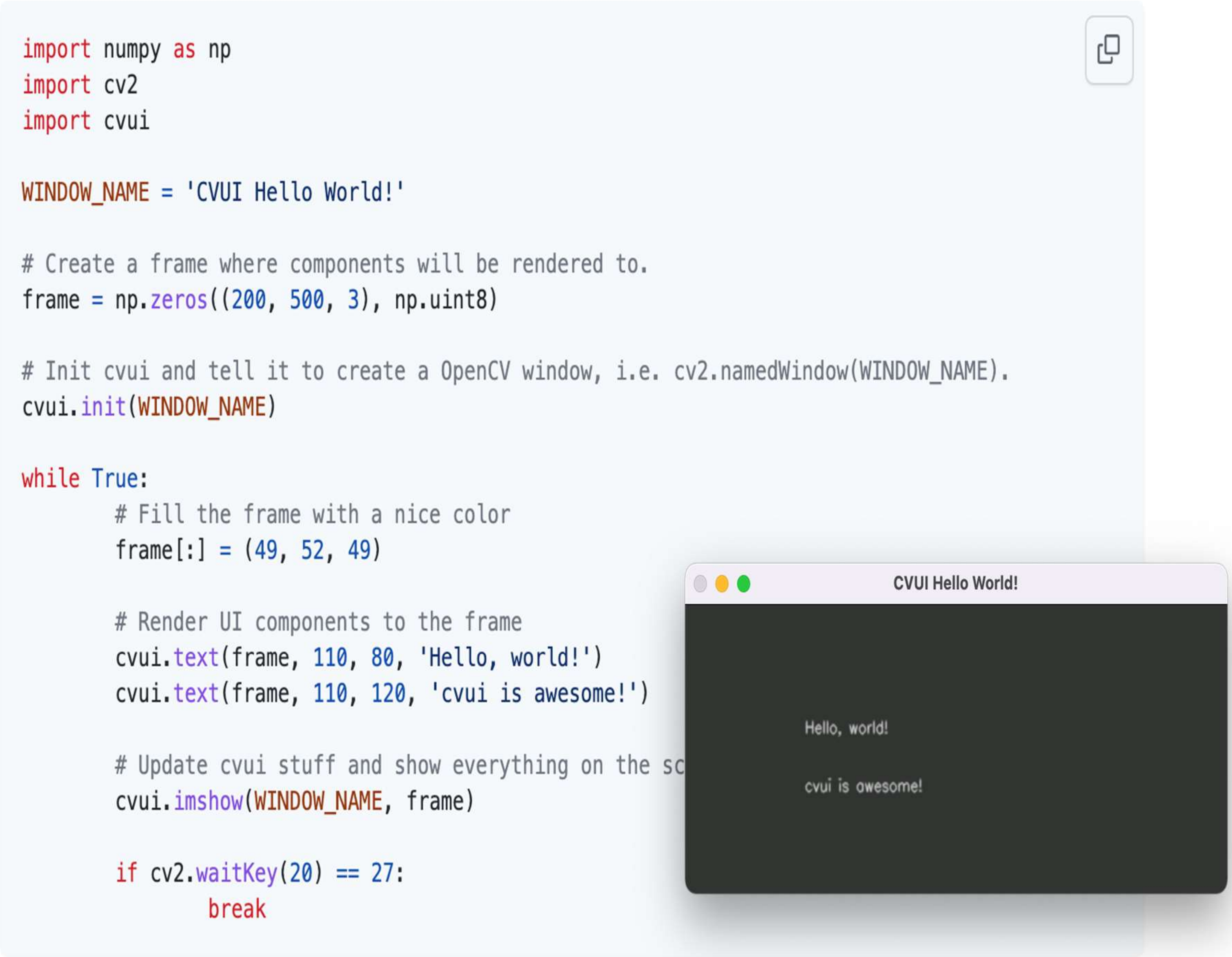


Figure 1: Button with auto-adjusted width.

Also, if you want to use python:

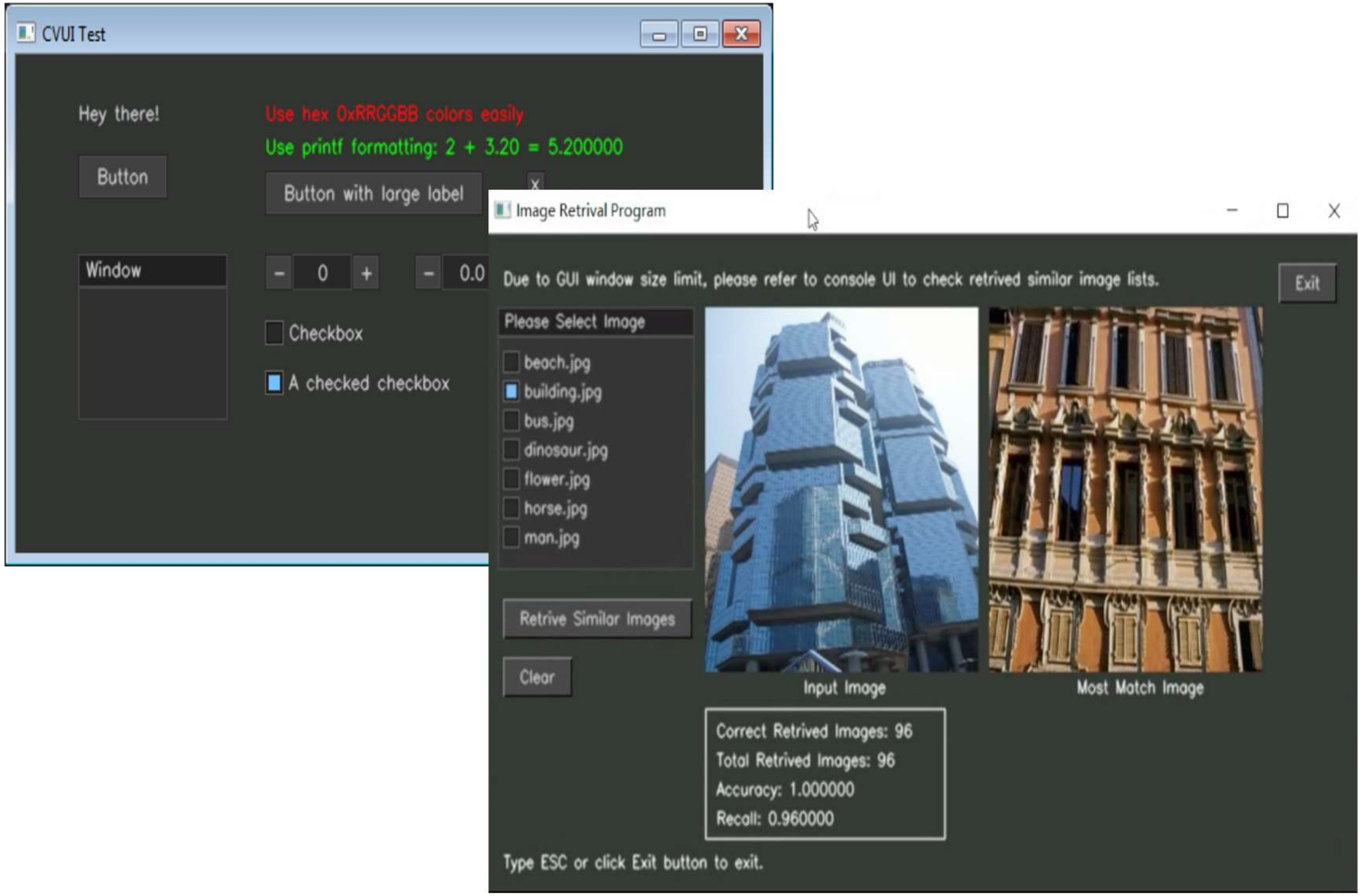


```python
import numpy as np
import cv2
import cvui

WINDOW_NAME = 'CVUI Hello World!'

# Create a frame where components will be rendered to.
frame = np.zeros((200, 500, 3), np.uint8)

# Init cvui and tell it to create a OpenCV window, i.e. cv2.namedWindow(WINDOW_NAME).
cvui.init(WINDOW_NAME)

while True:
        # Fill the frame with a nice color
        frame[:] = (49, 52, 49)

        # Render UI components to the frame
        cvui.text(frame, 110, 80, 'Hello, world!')
        cvui.text(frame, 110, 120, 'cvui is awesome!')

        # Update cvui stuff and show everything on the sc
        cvui.imshow(WINDOW_NAME, frame)

        if cv2.waitKey(20) == 27:
                break
```

CVUI Hello World!

```
Hello, world!

cvui is awesome!
```

More complicated and functional UI



Hey there!

Use hex 0xRRGGBB colors easily

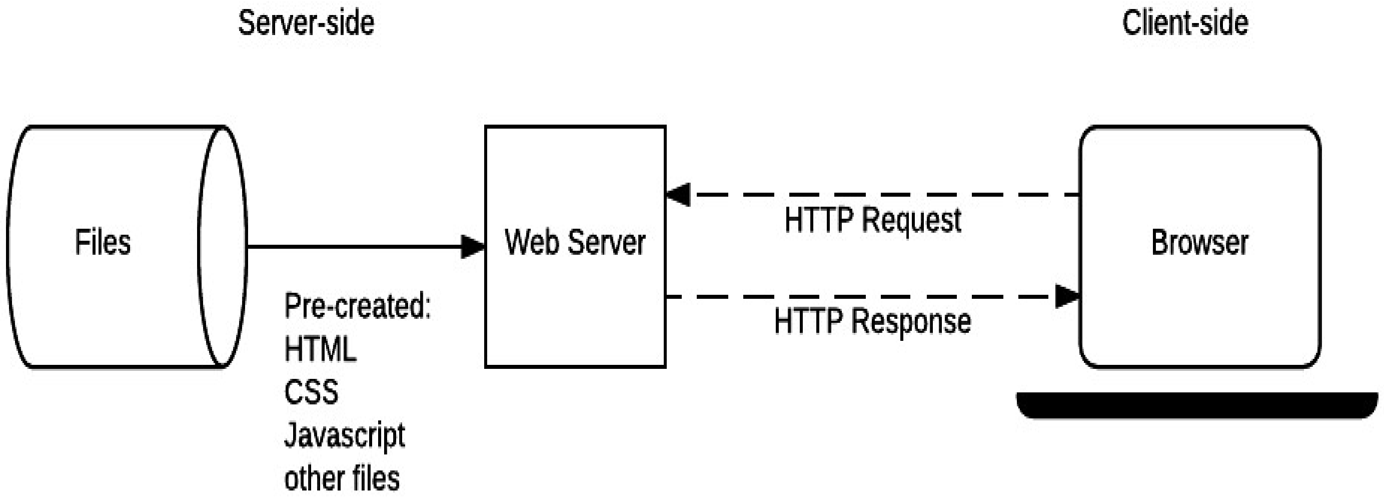
Use printf formatting: 2 + 3.20 = 5.200000

Due to GUI window size limit, please refer to console UI to check retrived similar image lists.

```
Correct Retrived Images: 96
Total Retrived Images: 96
Accuracy: 1.000000
Recall: 0.960000
```

Type ESC or click Exit button to exit.

WebUI: your program as a server



Webserver: your program

Brower: your UI

Files: your features, images, datasets, etc.

Web frontend UI

- Html and CSS
- Framework:
  - Jquery
  - Vue, React ……
  - Bootstrap

Advantages:

- More beautiful
- Easy to program
- More powerful
- No runtime dll needed

Some demo videos from previous years.

Thank you!

CS4185 Multimedia Technologies and Applications
Tutorial 4: Advanced Requirements