IEEE International Conference on Digital Signal Processing 2015

Teaching Large-Scale Image Processing over Worldwide Network Cameras

Wei-Tsung Su
Department of Computer Science and Information Engineering
Aletheia University
New Taipei City, Taiwan
au4451@au.edu.tw

Kyle McNulty and Yung-Hsiang Lu
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN, USA
{mcnulty, yunglu}@purdue.edu

***Abstract*—This paper presents a software system for large-scale image processing. Through this system, students may choose to analyze the images from several thousand network cameras deployed worldwide. This system allows both real-time analysis of live data or storing the data for off-line analysis. This system currently supports image processing using OpenCV-Python. The system allocates cloud instances as the computational engine and, as a result, allows users to analyze the images from many cameras simultaneously. The system demonstrates the ability to process 5,000 images from 500 cameras for lane detection in less than 2 minutes.**

Keywords—image processing; big data; DSP education; real-time

I. INTRODUCTION

Visual data has been growing rapidly in recent years. Cisco estimates that in 2018, a million minutes of video will cross the Internet every second [1]. It is common that a person has multiple cameras, in the mobile phone, in the tablet, and in the laptop. Users upload millions of images to social networks every day. The large amounts of images from many sources make the subject of image processing increasingly important. Many students are interested in learning the technologies of image processing. Commonly adopted approaches for teaching image processing use archival data. Students implement the processing algorithms in computer programs and execute the programs using some images provided by the instructors or the students themselves. It would be difficult if students want to analyze live data.

Many organizations deploy network cameras and stream the data to the Internet for anyone interested seeing. Figure 1 shows two examples of images from network cameras. Some network cameras provide video streams, while the others provide periodic snapshots (once every few seconds to every few hours). Through these cameras, students can observe many places in the world, for example, near their hometowns or the destinations of their next vacations. It would be educational if the students can analyze the data from the sources of their interest. Students may feel more motivated exploring new ideas if the data have personal meanings.

(a) (b)

Figure 1. Examples of cameras providing live data: (a) Yellowstone North Entrance. (b) Decorah Eagle Cam.

Organizing and analyzing data from network cameras, unfortunately, may be difficult. First, students have to find the cameras that provide data interesting to them. This can be achieved by searching the Internet. The result of these internet searches is typically a large set of dissimilar cameras. As a result, the student must be able to quickly sort and filter the cameras, to then manually verify that each camera is actually relevant to their interests. The next challenge is to retrieve the data from the cameras. Different brands of cameras may adopt different protocols and require different ways to retrieve the data. Even though many network cameras use HTTP, they have different paths to retrieve images using the GET commands. Moreover, the cameras may provide different data formats, such as JPEG, MJPEG, or MP4. Yet another challenge is to manage computational resources to execute analysis programs. Image processing can be computationally intensive. If a student chooses a dozen cameras and wants to analyze the data at a high frame rate, it is likely that the student's own computer will be

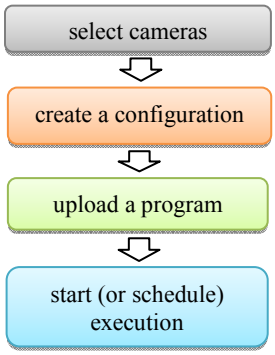


Figure 2. Procedure to use the proposed system.

unable to meet the performance requirements. Due to these many challenges, it is uncommon to ask students that are learning image processing to analyze live data streams from the sources of their interest. It is more common to give students image files.

We have been building a software system through which students can analyze the live data from many cameras without worrying about finding relevant image data or the underlying complex distributed system. Figure 2 shows the procedure to use this system. A student first selects the cameras of interest by their geographical location. The student can also see a snapshot from each camera to manually determine whether to choose this camera. It is possible to select a group of cameras in the same geographical region at once. It is also possible to select individual cameras. The set of selected cameras is part of a configuration. A configuration also includes additional information about the intended analysis, such as the desired frame rate (e.g. one frame every 10 seconds), the duration (e.g. two hours), and the starting time (e.g. from 8AM on Wednesday). Then the student can write a program (also called a module) that analyzes the live data and then upload the program to the system. This system currently supports Python because it is one of the most widely adopted languages [2]. Moreover, the OpenCV library is built-in to support image processing. The program must use the API (application programming interface) provided by the system. This API provides a uniform interface handling the data from heterogeneous cameras. If the student already has a program for analyzing data from files, only a few changes (mostly related to input and output) are needed. The system already has a dozen sample programs for students to learn how to use the API. Finally, the student creates a submission to execute the program; the student also has the option scheduling the execution for a later time. The system handles resource management by allocating cloud instances to execute the program. Multiple cloud instances may be allocated to meet the performance requirements.

This paper has the following contributions: First, we present a system that is designed to analyze images at large scales. Second, we explain how to manage heterogeneous cameras and retrieve data from these cameras. Third, we present several case studies showing how this system may be used to analyze live data. Fourth, this system already has alpha release to some users. More fifty people have registered as users.

II. BACKGROUND

Many studies have been conducted analyzing video and image. Kastrinaki et al. [3] write a survey paper about the techniques for processing video from traffic cameras. Some datasets are publicly available for example, ImageNet [4] and AMOS [5]. These are archival data and do not provide live data. Many organizations provide real-time data to the public. The departments of transportation in many countries deploy traffic cameras and make the data available. Some provide video streams, and the others provide frequent snapshots. Many national parks install network cameras in visitor centers. Many cameras are deployed in the United Kingdom observing waterways [6]. The data can provide valuable information about our environment but significant efforts are needed to use the data. Different methods are needed to retrieve data from different sources. Some cameras have web servers inside and have specific IP addresses. Even though these cameras support HTTP, different paths are needed to retrieve data from these heterogeneous cameras.

III. TEACGING LARGE-SCAKE IMAGE PROCESSING

In the proposed system, users can write and upload their programs to process and analyze the live images from worldwide network cameras. Moreover, cloud computing is utilized to process a large volume of images in a parallel manner. Thus, we believe that the proposed system is beneficial for teaching and learning large-scale image processing due to the following reasons.

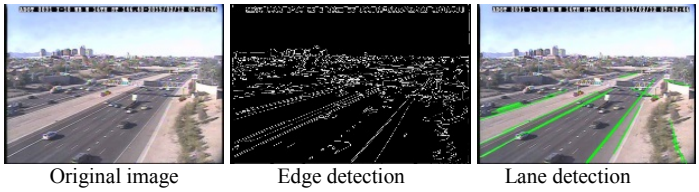
1. The proposed system supports the OpenCV library, which implements many algorithms for computer vision [7], such as feature extraction, edge and motion detection. Moreover, the OpenCV library has rich documentations (including sample programs and the description of the underlying algorithms, as well as citations of relevant publications) and large user community. As a result, the learning curve of image processing could be smoother.
2. Students can easily select a large set of live images from worldwide network cameras. For testing the adaptability of their analysis programs, students can select cameras which provide images in different environments (such as indoor vs. outdoor, daytime vs. night). Live images encourage students to develop better programs that handle any unknown image input, instead of programs that are tailored and designed for a specific set of image inputs.
3. Students do not need to worry about interfacing with cloud instances when their analysis programs process images from hundreds of cameras. The proposed system will automatically allocate cloud instances to execute the programs.

The system could help students learn developing application of image processing in three phases described as follows.

A. Design Phase

Image processing is widely used in many applications in recent years. Microsoft Kinect [8] demonstrates its wide diversity of such applications. Unfortunately, the learning curve of image processing is still too high.

For education purposes, the system allows students to learn from a top-down approach: they may use OpenCV and treat the same programs as black boxes first, with only tuning parameters. As shown in Figure 3, students could quickly write a lane detection program with Canny edge detection and Houge line dedection algorithms without undersing the details. After gaining some intuition, the students can then study the details of these algorithms for better insight.



Original image Edge detection Lane detection

Figure 3. Using computer vision algorithms to analyze images from traffic cameras.

B. Development Phase

A student can write a module to analyze images. The module may have the following classes:

1. The FrameMetadata class allows the student to obtain the information of a frame captured from cameras, such as date, time, and camera ID.
2. The CameraMetata class allows the student to obtain the information of a camera, such as latitude and longitude.
3. The Analyzer class allows the student to analyze the images from many sources.

The student's program needs to override in three methods:

1. The method initialize is called once at the beginning of the execution. The student can initialize tuning parameters used in the program.
2. The method on_new_frame will be called every time a new frame is retrieved from the selected cameras. This event handler is automatically triggered by the system. The image processing will take place inside this method.
3. The method finalize is called once after all frames are analyzed based on the configuration. The student can perform the final calculation (such as summarizing the information from all frames) and save results in files.

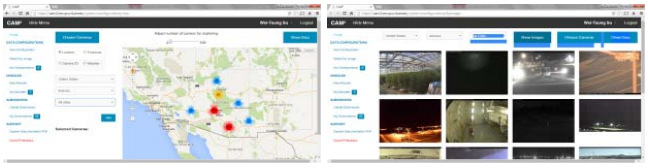


Figure 4. Web user interface for camera selection. (a) Select cameras by location, time zone, Camera ID, and weather. (b) Select cameras by image.

C. Testing Phase

As shown in Figure 4, the system currently allows students to select cameras by location, time zone, camera ID, weather, or by image. A set of images with different characteristics may be used to evaluate whether the student's program can handle the different environments (such weather and lighting). For example, Figure 5 shows images from different cameras, a desired program for lane detection should be able to handle these images.

IV. CASE STUDY: LANE DETECTION

Edge detection is common in teaching image processing. Students can apply the techniques in edge detection for detecting lanes in these examples. One purpose of lane detection is to determine whether any car spans two lanes and cause danger to the other vehicles. In the design phase, a student could find many examples for lane detection algorithm (LDA) in OpenCV [9, 10]. Some sample programs are available written in OpenCV-C# or OpenCV-Python. After studying these resources, the student can learn that LDA is composed of multiple steps, using Canny edge detection, Hough line detection, and line filter. In this phase, the student could also learn the effect of parameters of these algorithms, as shown in Figure 6.

Only a few changes (mostly related to input and output) are needed for migrating a program that processes archival data (i.e., files) to processing live data using the system's API. The tuning parameters of LDA are initialized in the initialize method. The main body of is in the *on_new_frame* method for processing the arrival of each frame from a camera. Finally, in the finalize method, the program summarizes the results, such as calculating the average from all frames. Figure 7 shows an example of an analysis program. To test this program, a configuration is created using 500 cameras in Arizona USA. One frame is captured and processed every 10 seconds from each camera. This is the desired frame rate. The actual frame rate can be lower due to network delays. The system processes more than 5,000 images in less than 2 minutes. Moreover, as shown in Figure 8, the results can be downloaded to evaluate wheatear LDA can handle the images with different features.

Figure 5. Test images in different environments

(a) Results of Canny edge detection with using 10 (left) and 100 (right) as the first threshold for the hysteresis procedure

(b) Results of Hough line detection with using 60 and 1 as the minimum length

Figure 6. Learning the effect of parameters of computer vision algorithms

```
class MyAnalyzer(Analyzer):
  def initialize(self):
    self.LaneCfg = {
      'canny'     : {
        'threshold1'   : 50,
        'threshold2'   : 150,
        'apertureSize' : 3
      },
      'houghlinesp' : {
        'rho'           : 1,
        'theta'         : np.pi/180,
      'threshold'     : 25,
        'minlinelength' : 60,
        'maxlinegap'    : 10
      },
      'filter'    :   {
        'invtheta'      : 180 / np.pi,
        'angle'         : 10
      },
    . . .
  def on_new_frame(self):
    # Get the frame
    frame = self.get_frame()

    # Get frame metadata
    frameMD = self.get_frame_metadata()

    # Get the date/time of frame
    date_time =
      frameMD.datetime.strftime(
        '%Y-%m-%d_%H-%M-%S')

    # Get the camera id
    camera_id =
      frameMD.camera_metadata.camera_id

    cfg = self.LaneCfg

    # Main body of LDA algorithm begin
    gray =
      cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # gaussian smooth
    gray = cv2.GaussianBlur(gray, (5,5), 3)

    # canny edge detection
    edge = cv2.Canny(
      gray,
      cfg['canny']['threshold1'],
      cfg['canny']['threshold2'],
      cfg['canny']['apertureSize']
    )
    # Hough line detection
    lines = cv2.HoughLinesP(
      edge,
      cfg['houghlinesp']['rho'],
      cfg['houghlinesp']['theta'],
      cfg['houghlinesp']['threshold'],
      minLineLength =
             cfg['houghlinesp']['minlinelength'],
      maxLineGap =
             cfg['houghlinesp']['maxlinegap'])
      . . .
    def finalize(self):
      . . .
```

Figure 7. Sample Lane Detection Program using the System's API

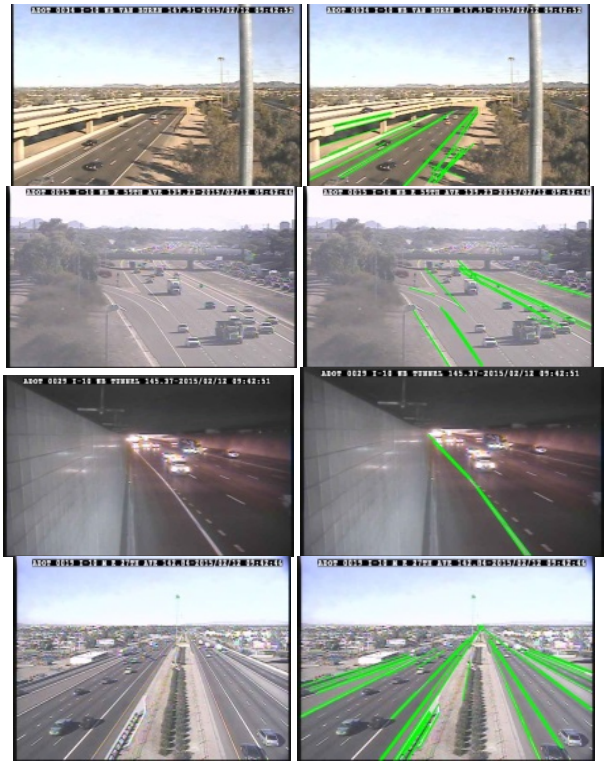


Figure 8. Results of LDA for handling images with different features

V. CONCLUSION

This paper presents a system which allows students to select cameras for image processing. The system retrieves live data from the selected cameras and students and experiment different processing methods using built-in functions in OpenCV or writing their own analysis modules. The system's API greatly simplifies the procedure for large-scale image analysis. This system is open to instructors and students as alpha users.

REFERENCES

[1] Cisco Visual Networking Index: Forecast and Methodology, 2013–2018

[2] Stephen Cass. Top 10 programming languages. http://spectrum.ieee.org/computing/software/top-10-programming-languages, Jul. 2014.

[3] V. Kastrinaki, M. Zervakis, , K. Kalaitzakis, A survey of video processing techniques for traffic applications, Image and Vision Computing, Vol. 21, No. 4, 1 April 2003, Pages 359–381

[4] http://www.image-net.org/

[5] http://amos.cse.wustl.edu/

[6] https://www.farsondigitalwatercams.com/

[7] http://openc v.org/

[8] Z. Zhang, Microsoft Kinect Sensot and its Effect, IEEE MultiMedia Magazine, Vol. 19, No. 2, 27 April 2012, Pages 4-10

[9] Leran Computer Vision, Lane Detection with OpenCV and C#. Available: http://www.learncomputervision.com/articles/programming/lane-detection-with-opencv-and-c/

[10] https://github.com/funningboy/ca