An Interactive Web-Based System Using Cloud for Large-Scale Visual Analytics

Ahmed S. Kaseb, Everett Berry, Erik Rozolis, Kyle McNulty, Seth Bontrager, Youngsol Koh, Yung-Hsiang Lu, Edward J. Delp
School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, USA

ABSTRACT
Network cameras have been growing rapidly in recent years. Thousands of public network cameras provide tremendous amount of visual information about the environment. There is a need to analyze this valuable information for a better understanding of the world around us. This paper presents an interactive web-based system that enables users to execute image analysis and computer vision techniques on a large scale to analyze the data from more than 65,000 worldwide cameras. This paper focuses on how to use both the system’s website and Application Programming Interface (API). Given a computer program that analyzes a single frame, the user needs to make only slight changes to the existing program and choose the cameras to analyze. The system handles the heterogeneity of the geographically distributed cameras, e.g. different brands, resolutions. The system allocates and manages Amazon EC2 and Windows Azure cloud resources to meet the analysis requirements.

Keywords: Visual Analytics, Interactive Website, Application Programming Interface, Network Camera, Cloud Computing

1. INTRODUCTION
What are the traffic conditions of the streets in New York City? Is it raining at the Eiffel Tower right now? Has the snow in a national park melted? Is a shopping mall crowded? These questions are frequently asked by city planners, meteorologists, and the general public. Although there are many image and video analysis programs for traffic monitoring, weather detection, and crowd monitoring, it is still hard to exploit thousands of online public cameras to find answers to these questions at any time. To answer such questions, we have constructed a system that can retrieve and analyze the live visual data from thousands of worldwide distributed cameras.

This paper presents CAM² (Continuous Analysis of Many CAMeras) as a system that addresses the following problems: (i) It takes significant effort to analyze images from thousands of cameras simultaneously. CAM² reduces that effort by providing an Application Programming Interface (API) that requires only slight changes to the existing analysis programs. (ii) Analyzing the data from thousands of cameras simultaneously requires significant amounts of resources. CAM² allocates and manages cloud resources to meet the computation and storage requirements. (iii) Cameras are heterogeneous, i.e. they have different brands and protocols for retrieving data. CAM² hides this heterogeneity so that the same program can analyze the data from different cameras.

This paper focuses on how to use the website and the API of CAM². A user can upload, execute, and download the results of analysis programs using the website by following this procedure:

1. The user can view an interactive world map with the geotagged cameras along with their recent snapshots.
2. The user can select the cameras to analyze using a variety of selection methods, e.g. country and timezone.
3. The user can specify the desired execution parameters, including the frame rate and the duration.
4. The user can upload the analysis program that uses the API of CAM². The API is event-driven: when new frames arrive, the analysis program is invoked. This event-driven API significantly simplifies the analysis program because it does not communicate with the heterogeneous cameras directly. The API requires slight changes to the existing analysis program because only the IO operations are modified.

Authors Emails: {akaseb, epberry, erozolis, mcnulty, sbontrag, koh0, yunglu}@purdue.edu, ace@ecn.purdue.edu
This paper has the following contributions: (i) To our knowledge, CAM$^2$ is the first interactive web-based system that enables users to analyze live images from thousands of cameras simultaneously. The system can be used for a wide range of applications. (ii) This system provides an API that makes it easy to migrate existing analysis programs with only slight changes. In particular, only the IO operations are modified. (iii) The system provides more than 65,000 worldwide distributed cameras discovered by our team. The system handles the heterogeneity of these cameras. (iv) CAM$^2$ executes the uploaded analysis programs on the data from the selected cameras at the specified frame rates and durations. To achieve that, the system allocates and manages cloud resources to meet the computation and storage requirements of the large-scale analysis.

2. RELATED WORK

Network cameras have been growing rapidly in recent years. Some websites$^{23}$ provide periodic snapshots from thousands of public cameras for various purposes. The AMOS (Archive of Many Outdoor Scenes) dataset$^4$ contains more than 650 million images from around 24,000 public outdoor cameras around the world. Since 2006, the images are captured several times per hour from each camera, and the dataset is publicly available. Hong et al.$^5$ developed a cloud-based framework for spatio-temporal analysis on large-scale distributed camera networks. Ramos-Pollan et al.$^6$ developed a cloud-based software framework for large-scale offline image processing and analysis. CAM$^2$ is different because it provides: (i) the live data from more than 65,000 cameras, (ii) the computing infrastructure to analyze the data, (iii) an API that requires slight changes to the existing analysis programs for a variety of applications, and (iv) a public website to use the system.

Many image and video analysis methods can extract useful information for a variety of applications, such as weather detection, human detection, anomaly detection, traffic monitoring, etc. Jacobs et al.$^7$ used the camera images of the AMOS dataset to estimate the weather conditions, e.g. wind speed. Dalal and Triggs$^8$ proposed an image analysis method to detect humans with different sizes, pose variations, and backgrounds. Kratz and Nishino$^9$ presented a statistical framework for detecting anomalous activities in crowded real-world scenes. Srivastava and Delp$^{10}$ proposed a video-based real-time surveillance system that can analyze the videos from traffic cameras (on highways/intersections). The system can extract important information about the vehicles, e.g. type, make, tire size, etc. The information can be used to detect anomalous events.

In our previous work, Kaseb et al.$^{11}$ introduced CAM$^2$ as a system for large-scale analysis of distributed cameras. Hacker and Lu$^{12}$ presented CAM$^2$ as an educational tool to teach students big data analytics. This paper is different from our previous work because it focuses in details on how to use the website and the API of CAM$^2$ to execute analysis programs for a variety of applications. This paper does not present new analysis methods. Instead, it uses OpenCV’s implementation of the background subtraction method proposed by KaewTraKulPong and Bowden$^{13}$ to illustrate how to migrate an existing analysis program to CAM$^2$.

3. OVERVIEW

CAM$^2$ is a system that enables users to execute computer programs analyzing the live visual data from thousands of cameras. Figure 1 shows the architecture of CAM$^2$. The website of CAM$^2$ allows the users to upload the analysis programs and select the cameras to analyze. The uploaded analysis programs have to be compatible with the API of CAM$^2$. CAM$^2$, as a cloud-based distributed system, handles the execution of the analysis programs and the communication with the cameras.

The cameras are deployed by various organizations, including departments of transportation, universities, companies, or individuals. The cameras are looking at a variety of scenes as shown in Figure 2(a), including streets, highways, classrooms, shopping malls, houses, natural scenes, etc. The cameras provide unprecedented amount of information that can help us understand the world better. The ultimate goal of CAM$^2$ is to bridge the gap between users with their analysis programs and the thousands of online public cameras. In other words, the goal is to enable the users to execute their analysis programs on the visual data from the cameras. To achieve that, CAM$^2$ has three main design goals: flexibility, ease of use, and scalability.
Flexibility: CAM² achieves flexibility by not assuming any prior knowledge about the analysis programs. CAM² can be used for a wide range of image analysis and computer vision applications, such as traffic monitoring, surveillance, weather detection, etc.

Ease of use: CAM² reduces the burden on the users. As shown in Figure 1, the users are responsible for only using the website and uploading the analysis programs. In addition:

1. CAM² handles the heterogeneity of the cameras. The cameras have different types: IP cameras whose IP addresses are known, and non-IP cameras for which some websites provide periodic snapshots. The cameras have different brands with different ways to retrieve the data. The cameras provide their live feeds in various formats, such as JPEG for images, and MJPEG or H.264 for videos. The cameras have also different resolutions and frame rates. CAM² handles this heterogeneity and the API provides a uniform way for the analysis programs to access the images from all the cameras.

2. CAM² handles the underlying computing infrastructure. CAM² uses cloud resources to meet the computation and storage requirements of the large-scale analysis. The users do not need to worry about cloud computing resource management. Instead, the users can focus only on their analysis programs.

3. The API of CAM² requires few changes to the existing analysis programs. In particular, only the IO operations are changed. This enables the users to migrate their analysis programs easily. More details about the API are discussed in Section 5.

Scalability: CAM² has to be scalable in order to analyze the tremendous amount of data from the thousands of cameras. CAM² handles scalability by using both private and public cloud resources. The system allocates and manages Amazon EC2 and Microsoft Azure cloud instances in order to meet the computation and storage requirements of the large-scale analysis. This system is designed such that the cloud instances communicate directly with the cameras without going through the server of CAM². This reduces the latency and enhances the scalability of the system.

4. CAM² INTERACTIVE WEBSITE

The website of CAM² is the users’ portal to the system. The users need to learn about only the website in order to execute their CAM²-compatible analysis programs on a large-scale. Through the website, the users can browse the cameras, select the cameras to analyze, set execution parameters (e.g. the analysis frame rate), upload their analysis programs, start and track the progress of their submissions, and download the analysis results.

The website of CAM² presents the cameras through an interactive map that shows a marker at the location of each camera as shown in Figure 2(b). The number of the cameras is large, and having a marker for each camera would cause the map to load slowly. Hence, the website groups the camera markers into clusters, each showing the number of the cameras in the cluster. The website shows a recent snapshot from a camera when its marker is clicked. CAM² periodically downloads camera snapshots and stores them locally in the server in order to reduce the latency of showing camera snapshots.

To execute an analysis program on a set of cameras, a user should upload the program and select the set of cameras to analyze. The user might need to execute the same program on different sets of cameras, execute different analysis programs on the same set of cameras, or execute the same analysis program on the same set of cameras.
Figure 2: Website Screenshots: (a) Snapshots from sample cameras in CAM² (b) Browsing the cameras through an interactive map and selecting the cameras for the analysis based on their locations, timezones, weather conditions, and IDs. (c) Browsing and selecting the cameras for the analysis based on their visual content (d) Setting the execution parameters of a configuration. (e) Browsing, editing, and erasing the saved configurations. (f) Browsing, editing, and erasing the uploaded modules. (g) Starting a submission by selecting a module and a configuration (h) Tracking the progress of, downloading the results of, and terminating the submissions.
of cameras at different times. To enable this flexibility, CAM$^2$ offers a three-step process for executing analysis programs as shown in Figure 3:

1. Create a **configuration** by selecting the desired set of cameras to analyze and setting some execution parameters, e.g., the analysis frame rate.

2. Upload a **module** which is a file that contains the source code of an analysis program. Currently, CAM$^2$ supports the modules that are written in Python and use OpenCV.$^{14}$

3. Start a **submission**, i.e., execute the analysis program in a selected module using the parameters of a selected configuration.

![Figure 3:CAM$^2$ three-step process for executing analysis programs](image)

In the next three sections, we present this three-step process. We show how this process enables users to execute analysis programs for a variety of applications. For more details, the website$^1$ of CAM$^2$ provides an online documentation and video tutorials illustrating the details about using the website and the API.

### 4.1 CAM$^2$ Configuration: Camera Selection and Execution Parameters

To create a configuration, a user should select the set of cameras to analyze and set some execution parameters, such as the analysis frame rate. Enabling the user to select a set of cameras for a variety of applications is indeed a challenge. In order to suit various applications, CAM$^2$ provides 6 ways of selecting cameras:

1. **Country, state, or city:** The user can select the cameras in a particular country, state, or city. This is essential if the user wants to analyze the data in a particular area, e.g., monitoring the traffic in Washington DC or Paris. Figure 2(b) shows a configuration using the cameras in New York City.

2. **Timezone:** The user can select the cameras in a particular timezone. This is useful if the uploaded analysis program has restrictions on the visual content from the cameras. For example, the analysis program might be designed to analyze images with high brightness, so the images should be taken during the daytime.

3. **Weather conditions:** The user can select the cameras whose cities have particular weather conditions (rain, wind, etc.). This is useful if the user wishes to execute a weather-related analysis program. For example, the user wants to execute a rain detection analysis program on the cameras that are likely to have rain. CAM$^2$ uses online weather services to retrieve the weather conditions of different cities.

4. **Camera IDs:** CAM$^2$ assigns a unique and fixed ID to each camera. If the user has a priori knowledge of the IDs of some particular cameras, the user can directly select these cameras using their IDs. The user can know the IDs of the cameras using the website.

5. **Camera map:** The user can select individual cameras though the interactive world map. When the user clicks a camera marker, the website shows a recent snapshot. If the snapshot is suitable for the user’s analysis purposes, the user can add the camera to the desired camera set.

6. **Visual content:** The user can select the cameras based on their visual content. The website can show a grid of recent snapshots from all the cameras in a selected country, state, or city. For example, Figure 2(c) shows the recent snapshots from the cameras in Antarctica. Then, the user can select the cameras that are suitable for the user’s analysis purposes.
The user can select the cameras using multiple methods. For example, the user might select the USA cameras that are likely to have rain, or the Eastern Time Zone cameras that are likely to have high wind speeds, etc. This can be beneficial for some computer vision applications, such as weather detection.

After selecting the cameras, the user should set the execution parameters which include: (i) the total duration of the analysis, (ii) the analysis frame rate, (iii) the limit on the number of cameras to analyze, and (iv) the number of past frames that are kept by CAM$^2$ and provided to the analysis program. This is essential if the analysis program needs to access old frames while processing new ones. Figure 2(d) shows a configuration that will be used to analyze 100 cameras for 2 hours at 1 frame per second. CAM$^2$ executes the uploaded analysis program for the specified duration using the specified frame rate on the selected cameras. The user might need to execute different analysis programs using the same configuration (i.e., using the same set of cameras with the same execution parameters). That is why CAM$^2$ saves the created configuration so that it can be used later. The user can browse, edit, or erase the existing configurations as shown in Figure 2(e).

4.2 CAM$^2$ Module: The Analysis Program

A module is a file that contains the source code of an analysis program. In Section 5, we discuss how to write a CAM$^2$ module in details. A user can write a module to monitor the traffic, detect the weather conditions, etc. The user might need to execute the same analysis program on different sets of cameras. For example, the user might need to monitor the traffic in New York City, and later decide to monitor the traffic in Paris. That is why CAM$^2$ saves the uploaded module so that it can be used later. The user can browse, edit, or erase the uploaded modules as shown in Figure 2(f). The website of CAM$^2$ provides a dozen pre-written modules. These modules have a variety of analysis programs, such as motion detection, moving objects detection, sunrise/sunset detection, etc. These modules are beneficial for the users who wish to try CAM$^2$ or to learn how to use it. These modules are also useful for non-experts and students to learn about image processing and computer vision.

4.3 CAM$^2$ Submission: Executing the Analysis Program

To start a submission, a configuration and a module should be selected as shown in Figure 2(g). Then, CAM$^2$ takes the responsibility of executing the analysis program in the uploaded module on the cameras and with the parameters specified by the selected configuration. The module can be a user module or a CAM$^2$ pre-written module. Figure 2(h) shows how the user can track the progress and the current state of the submission. As shown in Figure 4, the submission can be in one of the following states:

1. **Submitted**: When a user starts a submission, it remains in this state until CAM$^2$ starts allocating resources for the submission. The system starts allocating resources immediately after the submission is started.

2. **Allocating Resources**: The submission moves to the Allocating Resources state when the CAM$^2$ Manager module, that is deployed on the server, starts allocating cloud instances for the submission. Then, the CAM$^2$ Worker module, that is deployed on the cloud instances, starts executing the analysis program of the submission.

3. **Running**: When the Worker module starts executing the analysis program of a submission, the submission moves to the Running state. The submission remains in this state for the analysis duration specified in the configuration. The Worker retrieves the images from the selected cameras at the specified frame rate and invokes the analysis program. The website shows a progress bar to indicate the progress of a running submission as shown in Figure 2(h).

4. **Completed**: After the analysis program of a submission is executed for the specified duration, the running submission moves to the Completed state. The user can download the analysis results, and the website will provide a single compressed file that contains a directory for the results of each individual camera.

5. **Abnormally Terminated**: The uploaded module can have two types of errors: (i) errors that prevent the execution of the analysis program, such as syntax errors, API violations, or errors in the initialization stage of the analysis program. For this type of errors, CAM$^2$ terminates the running submission and moves it to the Abnormally Terminated state. (ii) runtime errors that might occur for only some frames. For both types of errors, CAM$^2$ includes the stack trace of the errors in the downloaded analysis results so that the user can fix the errors.
6. **Terminated**: The user can download the intermediate analysis results of a running submission. If the user is not satisfied with the analysis results for any reason, CAM\(^2\) allows the user to terminate the running submission. The system releases the cloud resources so that they can be used by other submissions. In this case, the submission is moved to the *Terminated* state.

![Figure 4: The life cycle of a CAM\(^2\) submission](image)

5. CAM\(^2\) EVENT-DRIVEN API

The API of CAM\(^2\) is event-driven, i.e. CAM\(^2\) invokes the analysis programs when new frames arrive. This event-driven design approach significantly simplifies the analysis programs. Analysis programs do not need to handle the underlying computing infrastructure and the heterogeneity of the cameras. Instead, the API of CAM\(^2\) provides a uniform way to analyze the data from all the cameras. Users can migrate their existing analysis programs to CAM\(^2\) easily because the API requires only slight changes to the existing programs.

The general structure of an existing analysis program can be divided into three main stages as shown in Figure 5(a): initialization, processing, and finalization. The first stage performs required initializations. The second stage reads, processes, and saves the results of the individual frames. The third stage releases the resources, computes and saves the overall results, etc. The operations performed by the analysis program can be categorized into: (i) IO operations that read the input frames (e.g. step 2 in Figure 5), or save the analysis results (e.g. steps 4 and 6 in Figure 5). (ii) non-IO operations that perform the actual analysis (e.g. steps 1, 3, and 5 in Figure 5). For most of the non-trivial analysis programs, the IO operations are usually significantly fewer than the non-IO operations.

5.1 The Analyzer Class: Event Handlers and IO APIs

The API of CAM\(^2\) provides the Analyzer class as the base for any analysis program class. The goals of the Analyzer class are to: (i) define how the analysis program class should be. Users should implement the *initialize*, *on_new_frame*, and *finalize* event handlers as shown in Figure 5(b). Table 1 shows more details about these event handlers. (ii) provide a uniform way for any analysis program to read the input and save the results as shown in Table 2. The users do need to worry about how to get the input frames from the heterogeneous cameras or how to manage the storage of the results on the cloud.

In order to migrate existing analysis programs to CAM\(^2\), users need to modify only the IO operations (e.g. steps 2, 4, and 6 in Figure 5). The IO operations are usually significantly fewer than the non-IO operations which remains the same (steps 1, 3, and 5 in Figure 5). Hence, CAM\(^2\) requires slight changes to the existing analysis programs. The following procedure should be followed to migrate an existing analysis program to CAM\(^2\):

1. Create a class that inherits from the Analyzer class.
Figure 5: The general structure of: (a) an existing analysis program, and (b) the corresponding CAM2-compatible event-driven analysis program. The dashed blocks represent the IO operations (i.e. reading the inputs and saving the results) which are modified to use the IO APIs of CAM2. The solid blocks represent the non-IO operations which remain the same.

Table 1: The event handlers provided by the Analyzer class. Users should implement these event handlers in order to migrate existing analysis programs to CAM2.

<table>
<thead>
<tr>
<th>Event Handler</th>
<th>Required?</th>
<th>Invocation</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialize</td>
<td>No</td>
<td>At the beginning</td>
<td>To perform initializations.</td>
</tr>
<tr>
<td>on_new_frame</td>
<td>Yes</td>
<td>For each new frame</td>
<td>To read, analyze, and save the results of the individual frames.</td>
</tr>
<tr>
<td>finalize</td>
<td>No</td>
<td>At the end</td>
<td>To release the resources, compute and save the overall results, etc.</td>
</tr>
</tbody>
</table>

2. Move the initialization, processing, and finalization stages of the analysis program to the initialize, on_new_frame, and finalize event handlers respectively as shown in Figure 5(b). Variables that are needed by multiple stages should be defined as object attributes.

3. Modify the IO operations of the analysis program such that they use the IO APIs of CAM2 as shown in Table 2. For example, use the get_frame method to read a new frame (step 2 in Figure 5), and the save method to save the analysis results (steps 4 and 6 in Figure 5).

The next subsection shows more details about how to migrate an existing analysis program to CAM2.

5.2 Migration Example: Background Subtraction

Figure 6(a) shows an existing background subtraction analysis program that uses OpenCV’s implementation of the method proposed by KaewTraKulPong and Bowden. The first stage initializes a background subtractor object (step 1). The processing stage reads the input frame (step 2), subtracts the background (step 3), and saves the input frame and its foreground mask (step 4). Steps 1 and 3 are non-IO operations, while steps 2 and 4 are IO operations.
Table 2: The IO methods provided by the Analyzer class. These methods are used to perform IO operations.

<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_frame</td>
<td>Input</td>
<td>To provide the most recent camera frames. If an old frame is needed, there is an optional parameter to specify the index of the frame.</td>
</tr>
<tr>
<td>get_frame_metadata</td>
<td>Input</td>
<td>To provide the metadata of the most recent frames. The frame metadata include the frame sequence number, the frame timestamp, and the metadata of the camera which the frame belongs to. The camera metadata include the camera ID, the latitude, and the longitude of the camera location.</td>
</tr>
<tr>
<td>save</td>
<td>Output</td>
<td>To save the results in a variety of formats, including text, or images.</td>
</tr>
</tbody>
</table>

```python
import cv2
import numpy as np
from analyzer import Analyzer

class MyAnalyzer(Analyzer):
    def initialize(self):
        bg_sub = cv2.BackgroundSubtractorMOG()

    def on_new_frame(self):
        frame = self.get_frame()

    frame_out = bg_sub.apply(frame)

    seq += 1
    cv2.imwrite('in_{}.png'.format(seq), frame)
    cv2.imwrite('out_{}.png'.format(seq), frame_out)

video_capture.release()
```

Figure 6: (a) Existing background subtraction analysis program. (b) The corresponding CAM2-compatible program. The solid blocks represent the non-IO operations which remain the same. The dashed blocks represent the IO operations which are modified to use the APIs of CAM2. For example, `self.get_frame` is used instead of `video_capture.read` to read a new frame, and `self.save` is used instead of `cv2.imwrite` to save the results. Note that steps 1-4 map to the corresponding steps in Figure 5.

Figure 6(b) shows the corresponding CAM2-compatible program. The event-driven programming model adopted by CAM2 makes it straightforward to map the initialization and processing stages to the corresponding event handlers. The non-IO operations (the solid blocks) remain the same, while the IO operations (the dashed blocks) are modified to use the APIs of CAM2. Initializing and releasing the video capture object are no longer needed because CAM2 manages the communication with the cameras. Figure 7 shows sample background subtraction results for a camera at Purdue University, USA.

The website of CAM2 provides a dozen pre-written CAM2-compatible analysis programs for moving objects detection, sunrise/sunset detection, etc. The variety of the available CAM2-compatible analysis programs emphasizes that the system is flexible and suitable for various applications. The migration example we presented and the examples available on the website show that CAM2 requires only slight changes to the existing analysis programs. Only the IO operations need to be modified to use the APIs of CAM2. If the users can organize the
non-IO operations in well-defined methods (e.g. the cv2.BackgroundSubtractorMOG method at step 1 and the apply method at step 3 in Figure 6), these methods can be migrated without any changes.

In our previous work, we used this background subtraction program to estimate the amount of motion in the cameras. CAM2 analyzed 2.7 million images from 1274 geographically distributed cameras over 3 hours simultaneously, using 1 frame every 5 seconds from each camera. The total size of all the images was 141 GB, and the data rate was 107 Mbps. The average resolution of the cameras was 768 \times 576. CAM2 used 15 Amazon EC2 cloud instances for this experiment. This large-scale experiment demonstrated the ability of CAM2 to analyze the visual data from thousands of cameras simultaneously providing useful insights about which cameras have high degrees of motion (deserve further investigation), and which cameras do not. We also used OpenCV’s implementation of the method proposed by Dalal and Triggs to detect and count the number of people in two cameras for 24 hours. This experiment showed that CAM2 is able to continuously analyze the cameras’ data for long periods of time. This is essential for many applications, such as traffic monitoring, weather detection, etc.

6. CONCLUSIONS

This paper presents CAM2 as an interactive web-based system that uses cloud for large-scale visual analytics. CAM2 enables users to execute image and video analysis programs on the live visual data from thousands of
cameras. The paper focuses on how to use the website and the API of the system. Using the website, the users can browse the cameras, select the cameras to analyze, set execution parameters, and execute the analysis programs. The API enables the users to migrate their existing analysis programs to CAM2 easily because only slight changes are required. The API of CAM2 is flexible such that it is suitable for a wide range of applications. The system is responsible for handling the heterogeneity of the cameras and the underlying computing infrastructure by allocating and managing Amazon EC2 and Windows Azure cloud instances.

7. ACKNOWLEDGMENTS

This project is partly supported by NSF IIA-1427808, CNS-0855098, and CNS-0958487. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF. We would like to thank the organizations that provide the visual data. The list is available at https://cam2.ecn.purdue.edu/acknowledgements.

REFERENCES