BeaverDam: Video Annotation Tool for Computer Vision Training Labels

by

Anting Shen

A project report submitted in partial satisfaction
of the Plan II requirements for the degree of

Master of Science

in

Computer Science

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor Kurt Keutzer, Research Advisor
Professor Trevor Darrell, Second Reader

Fall 2016

The project report of Anting Shen is approved.

Research Advisor Date

Second Reader Date

University of California, Berkeley

Fall 2016

Abstract

BeaverDam: Video Annotation Tool for Computer Vision Training Labels

by

Anting Shen

Master of Science in Computer Science

University of California, Berkeley

Professor Kurt Keutzer, Research Advisor

We present our annotation tool for frame-by-frame video bounding box annotation. The framework has been used in conjunction with Amazon Mechanical Turk as well as standalone, to annotate datasets for Berkeley Deep Drive and BMW. To accomplish this, we built upon ideas from previous works in this area, and we present our improvements and optimizations upon their user interfaces. We also introduce the idea of tuning such an annotation tool to reduce researcher's friction, which we argue is just as important as improving workers' workflow due to the high cost of researcher time. We share our experiences with existing tools, and our ideas (and code) for how to make the experience better for researchers. We hope our findings and contributions further reduce the cost of producing a labeled video dataset, and introduce ideas that will improve the quality of such software in the future.

Professor Kurt Keutzer
Research Advisor

Contents

Chapter 1

Introduction

Deep learning applications in recent years have come to require rapidly growing amounts of labeled training data. Often, accuracies can be boosted by adding data as much as by spending years on algorithmic development. For example, on the VOC07 benchmark, Faster-RCNN [1] with VGG-16 was able to eliminate 27.5% of errors in the much older R-CNN [2] backed by an equally old neural network architecture (mAP improved from 58.5 to 69.9). However, simply by including additional data from VOC12 and COCO, 29.5% of the remaining error was eliminated (mAP improved from 69.9 to 78.8).

Therefore, for real-world application development, data can be cheaper and more effective than scientists. While many existing tools support image classification – it is even built into Amazon Mechanical Turk (MTurk) – and some tools support bounding box labeling in images, few tools exist for frame-by-frame labeling in videos. VATIC [3] stands out as being one of the best, as not only does it make high quality annotations one of its main goals, but also cost and scalability.

My work borrows and improves upon many concepts and results from VATIC's user studies, but I focus on an additional goal that is extremely important in creating datasets for real applications. That goal is researcher happiness. Although VATIC extensively tested its "User Interfaces", I argue in chapter 2 that both the annotators and the experimenters are users, and the interfaces should be smooth for both when creating a tool.

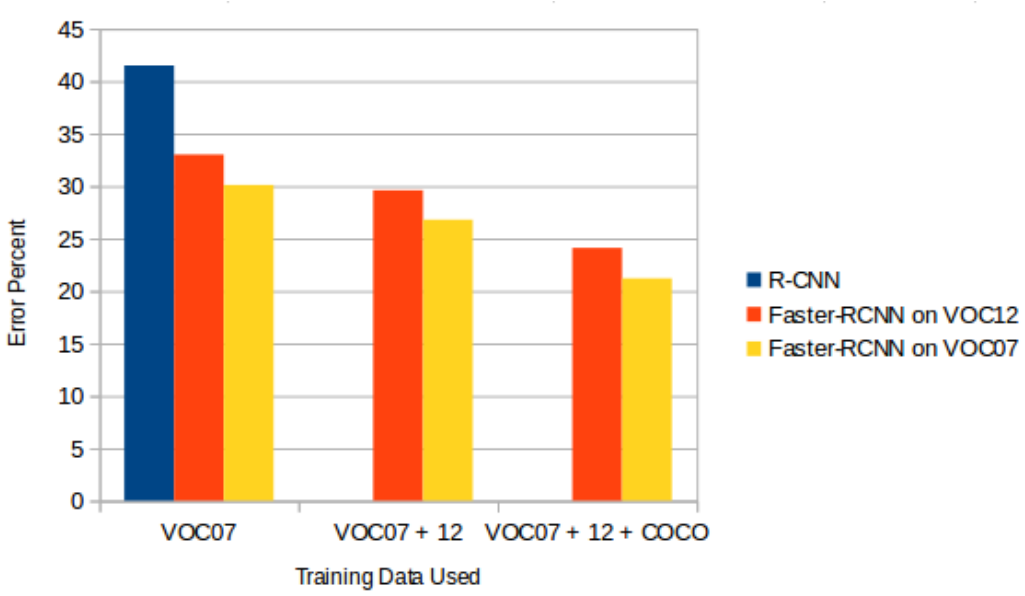

Figure 1.1. Improvement of error as data increases, compared to error improvements due to algorithm improvements. As we can see by comparing the drop from RCNN to Faster-RCNN on VOC07 only, and the drop from adding training data to Faster-RCNN, increasing data contributes significantly to error reduction.

Then, in chapter 3, I discuss my take on VATIC's User Interface principles for the annotator, and improvements upon them.

I also release all related code for BeaverDam, my video labeling platform, on Github.[1] At time of writing, the library has been used by Berkeley Deep Drive, DeepScale, and BMW.

Related Work

There are a large number of static image annotators available. For most computer vision applications that only require images instead of videos, these simpler alternatives should be used. For bounding box annotations, crowdsourcing sites such as Mechanical Turk or Crowdflower may have templates providing such functionality. For more complicated labels, tools such as LabelMe [4] provide many features for tasks such as image segmentation. However, none of these tools provide the ability to easily label every frame of a video.

[1] http://github.com/antingshen/beaverdam

There exists some public datasets of labeled videos. The YouTube-8M dataset [5] has millions of annotated videos, but they are annotated using the Inception V3 model trained on ImageNet. The KITTI dataset [6], as well as a dataset being built at Berkeley Deep Drive, have labeled videos. But these videos are not labeled every frame, instead an image is sampled every few seconds. Those images are then labeled using static image annotators. While these types of sampled image datasets are still useful, they don't allow research on supervised learning using nearby frames of a video, such as using an LSTM that uses each frame as input.

BeaverDam is designed to create frame-by-frame labeled video datasets. Such existing datasets include VIRAT [7], a dataset created using VATIC [3]. However, labeling videos with existing tools is difficult and expensive, so these datasets are rarer and smaller. Even setting up the first job is often a week-long project, consuming valuable researcher time. Modifying them for each dataset's custom specifications is also difficult. We created BeaverDam to address these problems.

Chapter 2

Experimenter Interface

During our investigation, we had many frustrating experiences with existing research tools in this area. These included installation issues, configuration issues, and dealing with unintuitive and undocumented interfaces. For example, there are dozens of open issues without solutions on the VATIC GitHub, and its installation script fails at multiple points on a brand new Ubuntu box. Due to preciousness of researcher's time, we believe the ability for researchers to test and iterate quickly is just as important as worker speeds when it comes to video annotation. Therefore we have placed improving the experimenter's experience as one of the main goals of this work.

2.1 Interface for researcher

For a basic user creating a crowdsourced video dataset using the default configurations we used, BeaverDam provides a streamlined interface. We provide a setup script that is tested on clean installs of Ubuntu 14.04 and Ubuntu 16.04. In comparison, VATIC was tested on an unspecified Ubuntu installation with exact Apache and MySQL installs, and while the install script claims that it should work on any operating system in theory, installation is difficult in reality. Additionally, our install script configures everything from Nginx and TLS to database config and backups. The user only needs to place their keys and

credentials in the locations specified in our documentation. While a containerization system such as Docker would have also solved VATIC's issues and ensure future compatibility, we felt that the additional complexity is not worth it, as many of our users in the research community are unfamiliar with Docker.

To use BeaverDam after installation, we provide a web interface for researchers to easily add and view videos and jobs. We feel that this is superior to VATIC's command line based approach, as the number of flags needed to specify various configurations was overwhelming. However, to allow experimenters to load large number of videos or perform other tasks programmatically, BeaverDam also provides a Python shell interface, backed by Django, to expose every functionality through Python.

Lastly, as BeaverDam is HTML5 video based, no frame extraction step is necessary. H.264 encoded videos will work without preprocessing. However, we do provide scripts to convert these videos to images with matching annotations to feed into machine learning tools if desired.

2.2 Decoupled modules

As the users of BeaverDam will most likely need to modify BeaverDam to fit their needs, we've emphasized modularity in our designs. Since modularity is something VATIC did well with their turkic, vatic, and pyvision splits, we've taken a similar approach. Our infrastructure is split between a crowdsourcing module, an annotation module, and a CV-based tracking module. But we decided to go a step further and make other parts of our platform replaceable as well.

To serve videos, we've provided Nginx to allow VATIC users to continue serving videos efficiently on the same server as their application. But to allow for scalability, users can use AWS S3 or any other CDN, or even a mix, by specifying so when creating jobs. Similarly, our setup pipeline is done through Ansible, a configuration management tool popular in industry. This allows users to deploy on their own servers, or in the cloud.

We also understand that not everyone wants to use Amazon Mechanical Turk. While past research has proven Mturk to be efficient and reliable, companies in industry seeking training data may prefer alternatives such as CrowdFlower, or may even choose to label in-house, or outsource to contractors. BeaverDam is designed with this flexibility in mind as it carries its own authentication and works independently of Mturk. This allows companies with the resources to seek cheaper labor in other countries to use BeaverDam as a platform for their workers and track progress and pay without the need for Amazon Mechanical Turk, which carries a 20% fee.

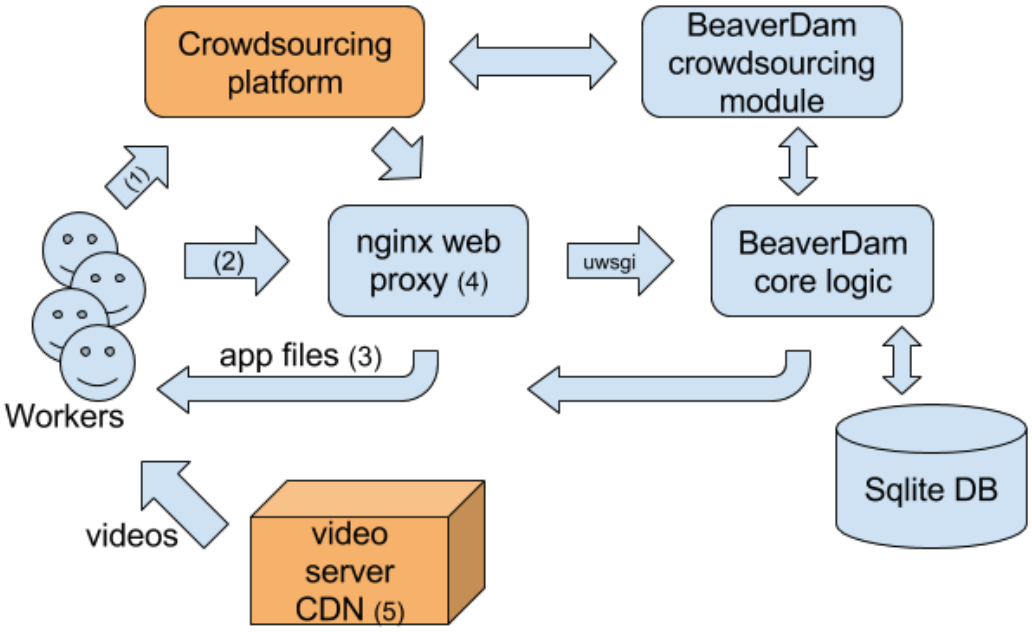

Figure 2.1. BeaverDam's backend server logic. The annotation app is sent in (3). Workers can either be hired through a crowdsourcing platform (1), or hired in-house and use BeaverDam directly (2). The web proxy (4) smoothly handles many requests and forwards static files, and performs HTTPS authentication with HSTS to meet Mturk security requirements. A video server or cloud provider CDN (5) is used to reduce worker download waiting times, a problem of other video labeling tools.

2.3 Patterns

To facilitate modification of our code, we adopted the Django framework because it sets fairly strict conventions on how a project is structured. Someone familiar with Django

will immediate know where to find the files responsible for each function of the backend. As there is a fair bit of frontend logic to video annotation, we've organized the code responsible according to well defined patterns as well. Under Our Pattern Language [8], the frontend follows the Model-View-Controller structure, and each of the three components is event-based. While using events to communicate within an application may seem like additional complexity at first, it enables decoupling and extends the existing events from user interaction.

2.4 Dependencies

The number one problem we encountered during our installations of VATIC was broken dependencies. Its GitHub issues point to problems users are having due to MySQL, cython, ipinfodb, and gcc versioning. BeaverDam addresses this problem in two ways. First, we greatly reduce the number of dependencies. Python 3 and two python packages installed through pip are the only dependencies required to run BeaverDam. Three other dependencies, Nginx, supervisor, and uwsgi, are recommended for efficient deployment. And to avoid the burdensome installation and configuration of a database that VATIC required, we use sqlite3, which is a Python standard library. In this case, it proves to be enough to handle even a fairly large amount of annotations, and can be swapped out easily if needed. Second, we version each dependency. This ensures that newer versions of dependencies in the future cannot break compatibility. For front-end libraries, we check-in the required files directly instead of using a package manager, avoiding extra complexity.

Another problem we encountered with VATIC's dependencies was its system-wide installation and configurations. Different users sharing a server would often leave bad state for others, causing hard-to-track bugs. We enclose the project in a virtual environment, and we include an idempotent script that verifies all required state, and fixing them if necessary. Our system is designed to be quickly set up on new VM instances, so one can perform upgrades and fixes by discarding the entire server VM and starting from scratch.

When choosing our dependencies, we strived to use the latest versions of mature tech-

nologies. We use Python 3.5 and Django 1.10, the latest versions available at the time. We also use the ES6 version of JavaScript, which includes many features to enhance readability and programmer happiness. While alternatives such as TypeScript provides additional features, ES6 is supported without a transpiler and avoids complexity. Browser compatibility of ES6 was a small issue with workers, but we expect this to improve as browsers fully adopt the standard and workers update to the latest version of their browsers.

2.5 Security

As BeaverDam aims to be industry quality software instead of research quality, security considerations are important. While one would not expect malicious users, it's important to follow security best practices as a preventative measure. Furthermore, having a full set of security tools allows the app to perform its own authentication, adding to modularity by allowing it to function as a standalone app without Mechanical Turk, unlike VATIC.

The first and best line of defense is regular backups. To back up data and configuration in VATIC, one had to perform a MySQL database dump and ensure all config files around the server are duplicated. In BeaverDam, configuration is checked into version control, and the database is in a single file for easy backup and restore thanks to Sqlite. Our setup tool automatically sets a daily backup to Amazon S3 if the app is configured with access keys. We also implemented other critical security features, such as TLS, HSTS, CSRF protection, and clickjacking protection. These features are automatically installed and activated by our setup script, with the user only needing to provide TLS certificates for their domain. [1]

[1] We recommend **letsencrypt** for easy and free TLS certificates.

Chapter 3

Annotator Interface

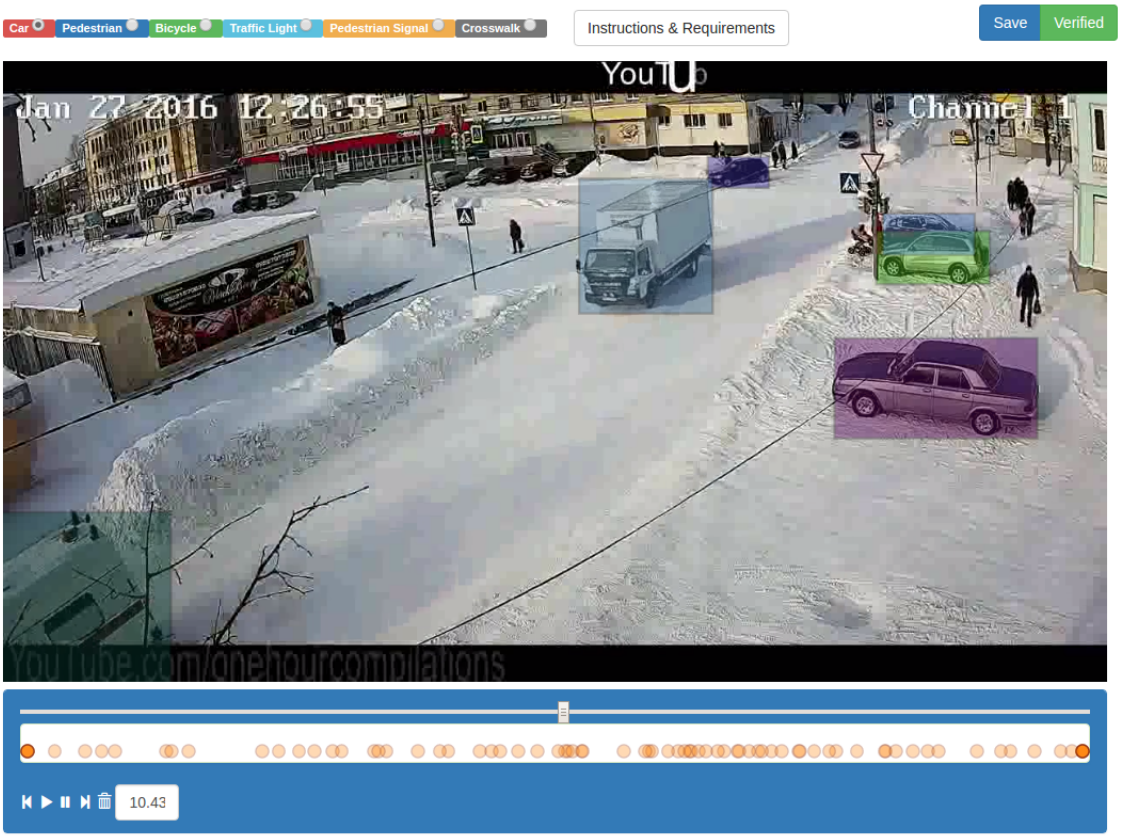

Figure 3.1. BeaverDam's expert verification interface with user keyframe scheduling enabled, border buffer disabled. The workers see a similar interface. Note the keyframe scheduling interface integrated with the video player, the single click multiple create interface, and objects being allowed to partially or wholly off-screen in the bottom left corner.

The productivity of annotators is the focus of most of the prior works in this area. During the development and deployment of BeaverDam, we've gathered extensive feedback from our turkers and testers. We also implemented features suggested by past works such as VATIC, and introduce several new ones to address issues found during the testing of VATIC.[1]

3.1 Keyframe scheduling

A big decision for a keyframe-based video annotation tool is whether the worker or the tool chooses the keyframe schedule. From past user studies, we know that human workers don't always choose the best keyframes, leading to inefficiencies. In fact, VATIC showed that by switching to a fixed keyframe schedule, workers were able to label faster. However, in our studies we found it to not always be the case. Since our videos contain a varied mix of stationary objects, slow & linearly moving objects, and fast objects with irregular motion, it makes sense to use fewer keyframes for easier objects. While a flexible keyframe schedule could potentially be automated using tracking, we instead optimized the interface for user created keyframes.

We added a keyframe viewer bar that displays all keyframes and keyframes for the currently selected object. The user can click on these keyframes to edit them, or insert new keyframes by editing the boxes when on a frame that's not a keyframe. We also included keyboard shortcuts to jump between keyframes for experts who need fine grained editing.

3.2 Video playback for maintaining identity

VATIC's video player was a rudimentary image changer as its purpose was to give workers context when tracking objects. As we wanted more flexibility and power for more advanced labelers, we incorporated a full fledged video player instead of using VATIC's system of using JavaScript to advance images of frames.

[1] Sceenshots of select issues in this chapter are taken from the demo video on VATIC's website, and are representative of what we encountered.

One huge issue we solved with this setup was video loading times. When deploying VATIC, we found users complaining about load times, and sometimes jobs would time out before the video loaded, especially when using high resolution images for frames. When seeking in the video, the images would flash to white if it's not loaded from cache quickly enough. By converting the app to use HTML5 video, we take advantage of its ability to stream, similar to YouTube and other sites can display the video without loading all of it. The annotator can then begin annotating as the video loads. The playback is also much smoother, with a minor tradeoff of slightly slower rewinds, as videos are encoded in a way optimized for forward playback.

3.3 Reducing clicks

In addition to measuring time it takes for annotators to perform certain actions, we sought to minimize the number of clicks necessary. We find this to be a good rule of thumb when we don't have enough users to properly A/B test design choices.

First, we set the user in create/edit hybrid mode by default. A new object is created if the user drags in an empty spot, and an existing object is edited otherwise. VATIC requires the user to click a "create" button, which enables the user to annotate heavily overlapped objects more easily. We propose that it's more efficient to force a user to perform the extra steps of creating a box in an empty spot and dragging it onto the object in case of overlap, as heavy overlaps turned out to be rare in our datasets, so it was better to optimize for the common case.

Additionally, instead of having the user choose the type of object each time, we default to the most common label (car) or the user's last selection. This reduced clicks, but did result in more errors for new users who didn't notice that they must change the label from the default when labeling non-cars. While VATIC's prompting each time prevented this, we elected to solve it through a better tutorial to save time for trained annotators.

Finally, BeaverDam includes extensive keyboard shortcuts, with the aim of eliminating any need for the mouse aside from drawing and moving boxes. Tasks such as label selections

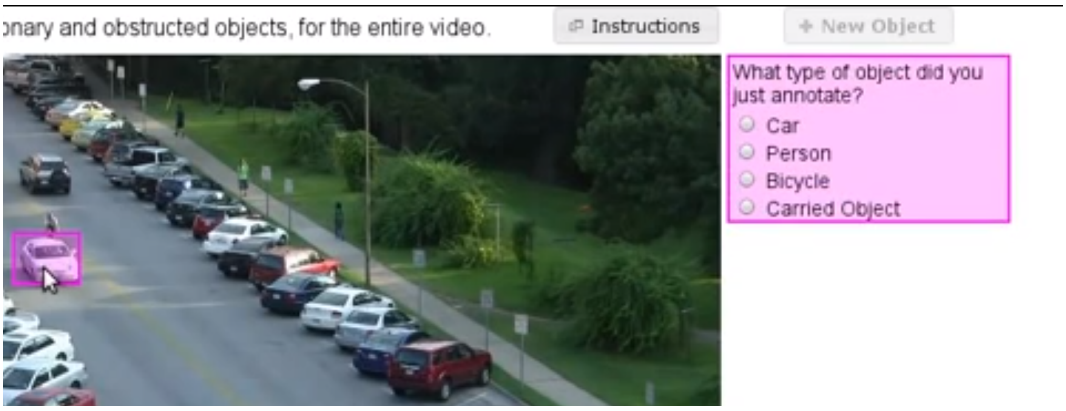

Figure 3.2. Explicit object creation in VATIC. BeaverDam guesses whether the worker wants to create a new object or edit an existing one by where they click, which is efficient in the general case. We also eliminate this additional prompt of asking for the object type and assume the type is same as the last created type by default.

and video playback are all controlled through the keyboard. We left in optional mouse controls, but it may be a good choice to remove them to enforce keyboard shortcut usage. Surprisingly, we have had turkers report that they were on tablets, and keyboard shortcuts were inefficient for them. However, we find in our small sample of annotators that tablet users are few and are less efficient in general.

3.4 Handling frame exit/enters

During our user tests, we found that users have the most difficulty when objects are entering or exiting the frame, as it's difficult for linear interpolation to handle a box whose size is increasing but one edge is fixed to the edge of the frame.

This was one of the main challenges when annotating using VATIC, as we had many driving videos where objects frequently entered and exited. To address this issue, we introduced two solutions. First, we allow boxes to be partially or wholly located outside the frame. This enables the user to guess at the true location of the object if the frame were bigger, which allows for much better linear interpolation. Then, we add a large padding around the border of the frame, essentially enlarging the frame with blank space, which makes drawing and editing boxes that are located partially outside the frame much easier.

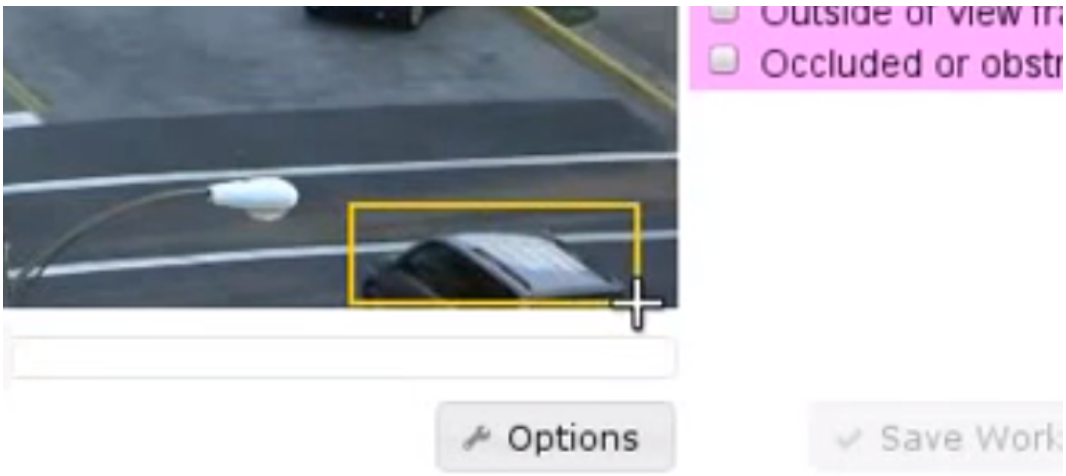

Figure 3.3. VATIC annotation of object entering frame. Note the user must draw the box exactly to the edge of the screen as shown here, as drawing fails otherwise. BeaverDam allows the user's mouse to draw to outside the frame, cropping back to the frame edge in post processing.

After labeling, any boxes that are partially outside the frame can be cropped to the edge of the frame.

Chapter 4

Conclusion

We have presented numerous improvements to existing video annotation tools. We first argued that optimizing for researcher time is as important, if not more important than optimizing for worker efficiency, and showed our contributions in that area. Then we demonstrated several ideas that allow for easier annotation by the workers that we discovered in our experiments. We showed how BeaverDam enables cheaper annotations to create large datasets that improve model accuracies in production.

As BeaverDam is designed to be extensible, we hope that the platform will adapt to new types of annotations. BeaverDam's modular designs shine best when new types of data, such as point clouds from LIDAR, require human labeling in the future.

References

[1] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[2] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Computer Vision and Pattern Recognition*, 2014.

[3] C. Vondrick, D. Patterson, and D. Ramanan, "Efficiently scaling up crowdsourced video annotation," *International Journal of Computer Vision*, pp. 1–21, 10.1007/s11263-012-0564-1. [Online]. Available: http://dx.doi.org/10.1007/s11263-012-0564-1

[4] B. Russell, A. Torralba, K. Murphy, and W. T. Freeman, "Labelme: a database and web-based tool for image annotation," in *International Journal of Computer Vision*, 2007.

[5] S. Abu-El-Haija, N. Kothari, J. Lee, P. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan, "Youtube-8m: A large-scale video classification benchmark," *CoRR*, vol. abs/1609.08675, 2016. [Online]. Available: http://arxiv.org/abs/1609.08675

[6] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[7] S. Oh, A. Hoogs, A. Perera, N. Cuntoor, C.-C. Chen, J. T. Lee, S. Mukherjee, J. Aggarwal, H. Lee, L. Davis, E. Swears, X. Wang, Q. Ji, K. Reddy, M. Shah, C. Vondrick, H. Pirsiavash, D. Ramanan, J. Yuen, A. Torralba, B. Song, A. Fong, A. Roy-Chowdhury, , and M. Desai, "A large-scale benchmark dataset for event recognition in surveillance video," in *IEEE Comptuer Vision and Pattern Recognition (CVPR)*, 2011.

[8] K. Keutzer and T. Mattson, "Our pattern language (opl): A design pattern language for engineering (parallel) software," 2009.