Driving dataset in the wild: Various driving scenes

Kurt Keutzer
Byung Gon Song
John Chuang, Ed.

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2016-92
http://www.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-92.html

May 13, 2016

Driving dataset in the wild: Various driving scenes

by Byung Gon Song

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Kurt Keutzer
Research Advisor

(Date)

* * * * * * *

Professor John Chung
Second Reader

(Date)

Abstract

For our dataset, we tried to collect data that includes accidents as well as diverse conditions described below. For data acquisition, we used YouTube to collect videos that have a Creative Commons License. For deduplication, first we split the videos into scenes, and then we used fingerprinting methods. Lastly, we created annotations on some of the scenes using Video Annotation Tool from Irvine, California (VATIC) software. The goal was to annotate every instance for each class: 'cyclist', 'van', 'tram', 'car', 'misc', 'pedestrian', 'truck', 'person sitting', and 'dontcare' as KITTI has done.

Contents

Chapter 1

Introduction

1.1 Background

A large and diverse training data set can improve the accuracy of deep learning. To train a model for object detection in autonomous driving, the KITTI dataset [1] is used in most research; however, KITTI is a relatively small dataset. And it is towards many visual elements such as time of day and camera orientation; one needs only to look at few images from the data to recognize them as belonging to the KITTI dataset [6]. The problems we are attempting to solve from this dataset are to reduce its bias, increase its size, and add instances. Also, we believe our dataset can improve object detection on the KITTI test set by 5% (Moderate) and 10% (Hard) by providing training data that are hard to recognize. In this paper, we present how to create such unbiased dataset with annotations.

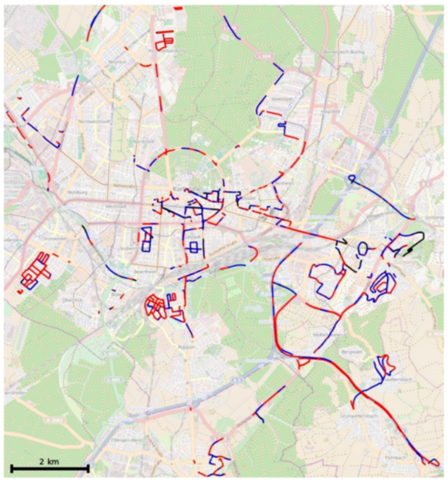


Figure 1: Recording zone. This figure shows the GPS traces of recordings in the metropolitan area of Karlsruhe, Germany. [3]

1.2 Problems Addressed

The improvements in our dataset include reducing its bias, improving its size, and adding examples. First, we tried to reduce its bias by collecting data from YouTube. Videos on YouTube are recorded under different lighting conditions, in different locations, and with different cameras. We also captured abnormal driving scenes that other datasets such as KITTI do not contain. We tried to include various driving scenes in unconstrained conditions. Our random YouTube sampling method also established sampling validity [2].

Second, we tried to create a larger dataset than the KITTI dataset by providing more training images. The KITTI dataset uses only about 1 hour of video as training data. In our proposed dataset, we collected about 200 hours of video and provided annotations.

Lastly, we tried to create a dataset that provides more instances that are hard to recognize. The KITTI dataset defines small or occluded instances as “hard” instances. We tried to obtain many of these samples in our proposed dataset.

Chapter 2

Data Acquisition Methodology

1.1 Sources

YouTube is the main source for our proposed dataset. We chose it as our main source because we wanted to collect driving videos and to create images based on the videos and YouTube had the largest body of videos to select from. We used the following search terms to collect videos: “car crash compilation,” “cyclist dashcam,” “dashcam video,” “motorcycle dashcam,” “pedestrian car accidents,” and “pedestrian’s compilation.” We used these terms to collect videos that contain accident scenes recorded from the driver’s point of view, as does the KITTI dataset. We also only collected videos that had a Creative Commons license.

1.2 Methodology

First, we used the `BeautifulSoup` Python package to download the URL. This Python package allowed us to get URLs of videos in YouTube. Once the URL was downloaded in text file, `youtube-dl` was used to download the actual video without the sound content. Before we downloaded the video, we removed the URLs that appeared several times in the text file or that had a YouTube Standard license.

1.3 Results

We downloaded videos with a Creative Commons License only. Before we de-duplicated videos, we collected about 2,084 videos adding up to 152 GB.

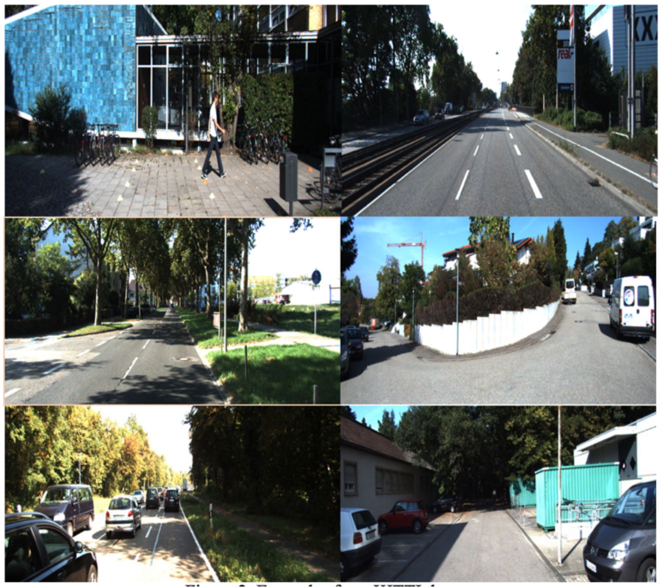


Figure 2: Examples from KITTI dataset

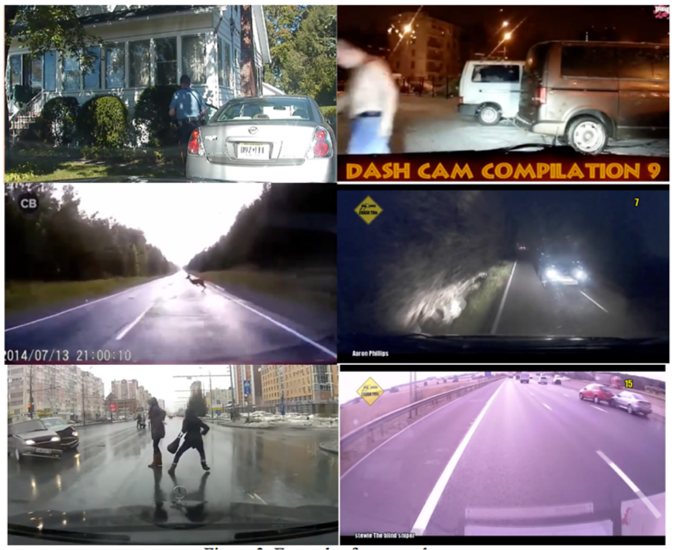


Figure 3: Examples from our dataset

Sample images in Figure 2 and Figure 3 show differences between the KITTI dataset and our dataset. As one can see, the size and the quality of images in the KITTI dataset are exactly the same, and they were recorded only during the daytime. Figure 3 shows that our dataset consists of videos that were recorded at different times, in different locations, and with different cameras. Also, our dataset has regular driving scenes, and additionally it has many abnormal scenes as seen in Figure 3c and 3e. Many of our videos have crash scenes, as the two images in Figure 3 show.

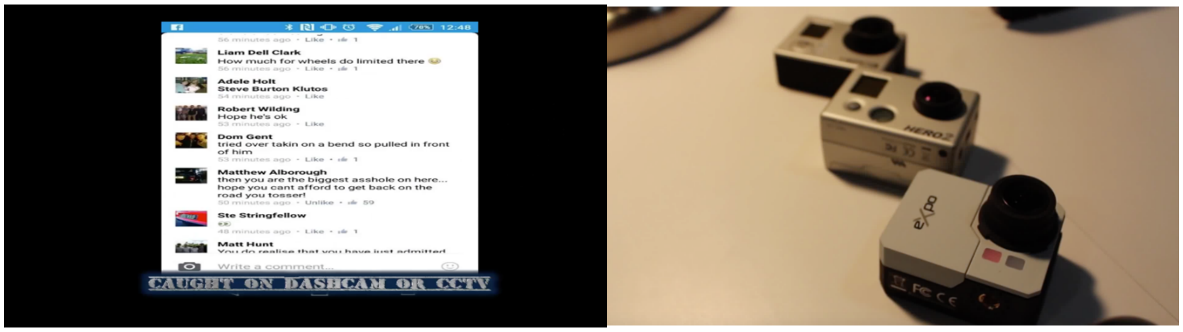

78% 12:48
56 minutes ago · Like · 1
Liam Dell Clark
How much for wheels do limited there
56 minutes ago · Like · 1
Adele Holt
Steve Burton Klutos
54 minutes ago · Like
Robert Wilding
Hope he's ok
53 minutes ago · Like
Dom Gent
tried over takin on a bend so pulled in front of him
53 minutes ago · Like · 1
Matthew Alborough
then you are the biggest asshole on here... hope you cant afford to get back on the road you tosser!
50 minutes ago · Unlike · 59
Ste Stringfellow
48 minutes ago · Like · 1
Matt Hunt
Write a comment...

Figure 4: Completely irrelevant images

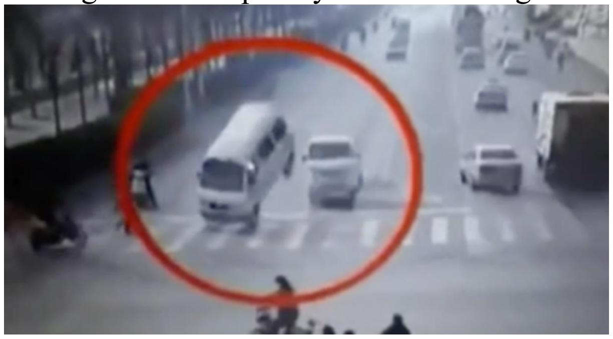


Figure 5: Unrealistic mark on image

The main problem with our dataset is that some videos contain totally irrelevant images, as Figure 4 shows. Another problem is that, as Figure 5 shows, some videos have marks drawn on scenes where accidents occurred. There will be more discussion about this in the **Future Work** section, but currently there is no good way to remove these videos from our dataset without manually filtering them out.

Chapter 3

Deduplication of Data

1.1 Problems Addressed

Since we collected videos that had Creative Commons license, other users were free to include these videos in their compilation, and as a result many video segments were duplicated in multiple videos. One of the goals for our proposed dataset is to create annotations on instances using Amazon Mechanical Turk; however, it would be a waste of our resources to annotate videos that are essentially identical. Therefore, we first removed duplicated contents from our dataset using the fingerprinting method. The fingerprinting method uses images as hash values to check whether videos are identical. We explored parallel process for this method.

1.2 Scene splitting

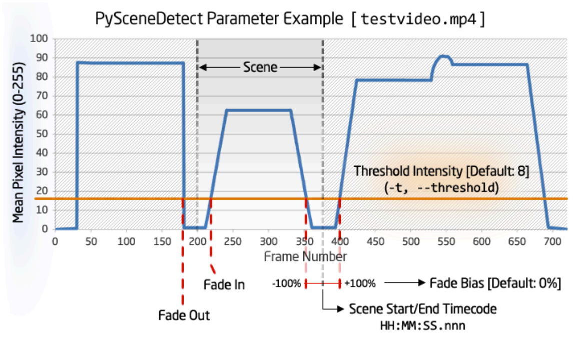
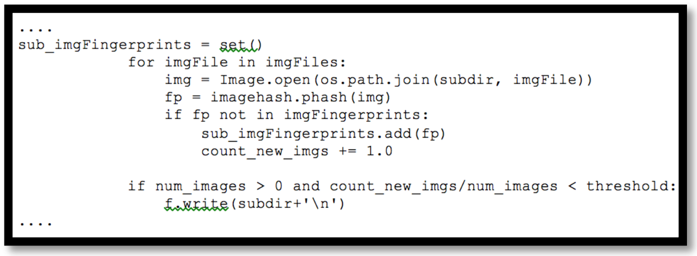


Figure 6: Visual example of the parameters used in threshold mode for `PySceneDetection` [4]

In order to remove duplicated contents from our dataset, we needed to split the videos first and remove replicated videos. We used `PySceneDetection` to slices videos into a several scenes. This Python package detects split points in a video, as shown in Figure 6. With given parameters, the algorithm detects the frame of a video where the mean pixel intensity of an image changes. Before we used the package, we converted frames per second of all videos to 20 and resolution was set at 800 by 600. We used `ffmpeg` for conversion and stored them to `videos-normalized` directory. Once we sliced the videos into smaller clips of the videos, we were had 58,042 scene files total, or 85.17 GB of files. The total duration was 201.22 hours. They are stored in six different folders. More details can be found in `README`.

1.3 Fingerprinting Method

Once we split the videos, we used a fingerprinting algorithm to determine the duplicated scenes. Each scene was divided into a series of images, and the fingerprinting algorithm for each image was converted into a hash value and checked to determine whether or not the value existed in the `fingerprint.p` file. The `phash` package converts an image into a hash value. It creates a hash value that is robust enough to deform the image. It does not create a different hash value for different pixel values of the same images. Using the package, we implemented an algorithm that counts how many images exist in the `fingerprint.p` file already; if it has too many similarities, then it is not included in our dataset.



```
....
sub_imgFingerprints = set()
            for imgFile in imgFiles:
                img = Image.open(os.path.join(subdir, imgFile))
                fp = imagehash.phash(img)
                if fp not in imgFingerprints:
                    sub_imgFingerprints.add(fp)
                    count_new_imgs += 1.0

            if num_images > 0 and count_new_imgs/num_images < threshold:
                f.write(subdir+'\n')
....
```

Figure 7: Simple fingerprinting algorithm for deduplication of video

1.4 Parallel Process

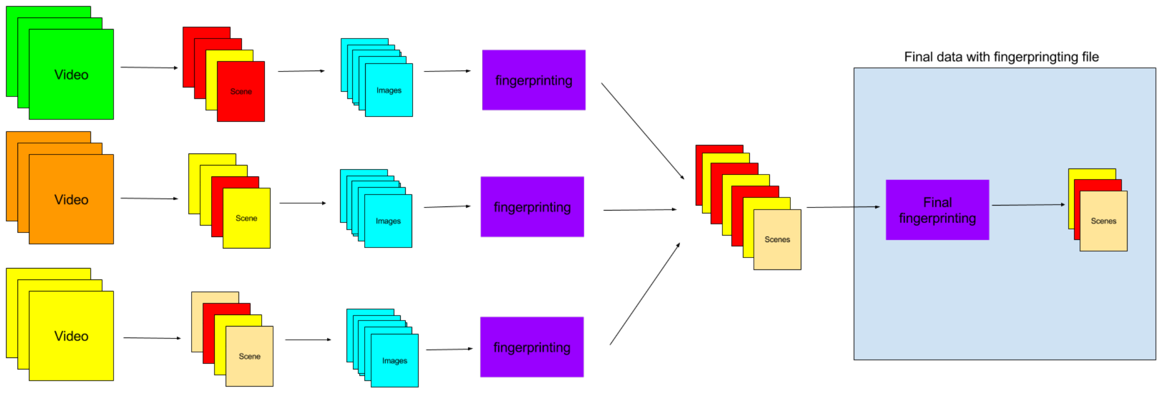
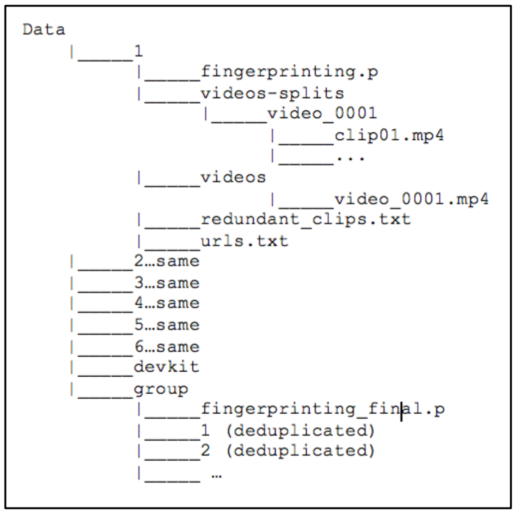


Figure 8: Scalable parallel deduplication process

This deduplication process can be done in parallel, and it is also scalable, as Figure 8 shows. It is parallel because the de-duplication process for subsets of video can be done within the videos in the subset. Once the de-duplication process was done on these subsets, they were reduced and the fingerprint algorithm method was repeated on these videos. Lastly, it is scalable because creating a bigger dataset with new videos does not require comparing them to other videos. They can just use the final fingerprinting method and in that way extend the dataset.

We developed the software architecture and implementation of our solution using the pattern-based approached described in [10]. The most relevant structural pattern for our parallel deduplication process is MapReduce, where computations are mapped onto independent data sets. During the process, we first distributed videos images, and mapped the computation of hash values of images. Computation in the fingerprinting process is a comparison between hash values of images and values in a fingerprint file. The most relevant computational pattern is the graph traversal computational pattern. At a reduced stage, the results of the map stage are summarized.

1.5 Results



```
Data
    |_____1
          |_____fingerprinting.p
          |_____videos-splits
                |_____video_0001
                      |_____clip01.mp4
                      |_____...
          |_____videos
                      |_____video_0001.mp4
          |_____redundant_clips.txt
          |_____urls.txt
    |_____2…same
    |_____3…same
    |_____4…same
    |_____5…same
    |_____6…same
    |_____devkit
    |_____group
          |_____fingerprinting_final.p
          |_____1 (deduplicated)
          |_____2 (deduplicated)
          |_____ …
```

Figure 9: Detailed description of final dataset

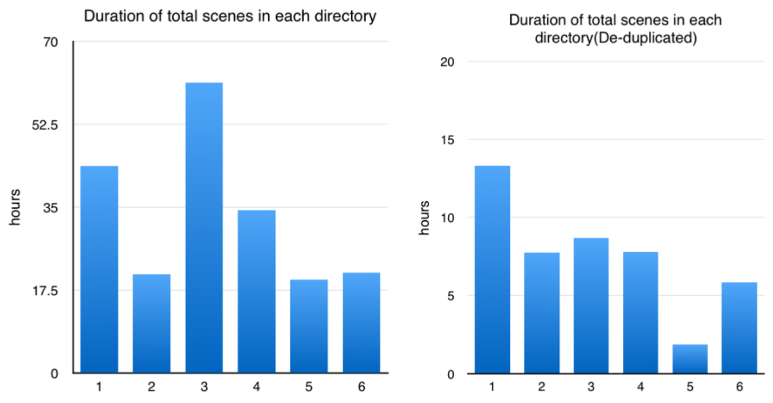


Figure 10: Histogram for durations of all videos in each directory (Left: Before deduplication; Right: After deduplication)

Figure 9 shows the final dataset structure. After removing duplicate contents in our dataset, we were able to create a dataset of 23 GB in size and 45 hours in length with ~20,054 clips of video. Our dataset contained irrelevant videos such as shown in Figure 4. We manually watched over 40 hours of video and removed irrelevant videos because there is no good way to remove these videos automatically. We additionally removed the following content:

1. Videos with really low resolutions.
2. Videos with too little variation. For an example, objects in video are not moving.
3. Distorted videos.
4. Videos displaying multiple scenes from different camera at the same time.
5. Videos with too low or too high viewing angles.

Chapter 4

VATIC

1.1 Introduction

For annotation, we used the VATIC (Video Annotation Tool from Irvine, California) software tool [11]. VATIC is a free video annotation tool for computer vision research that can run on Amazon Mechanical Turk.

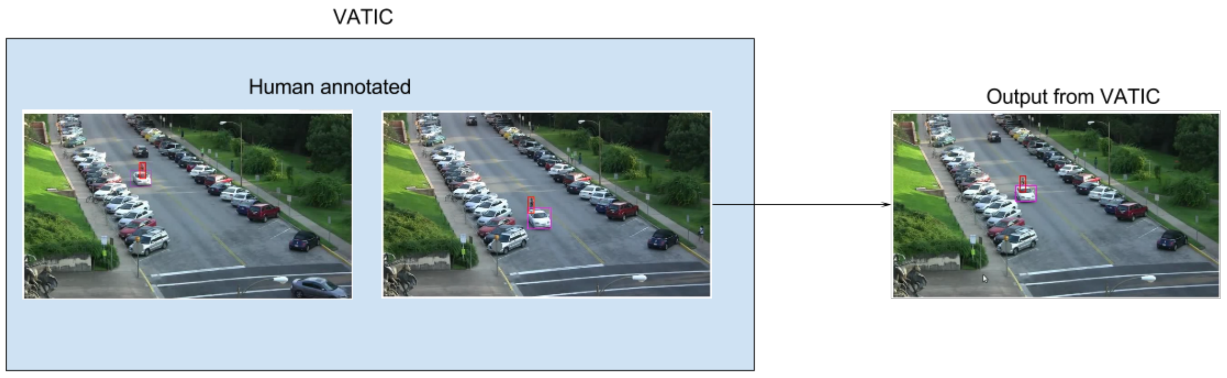


Figure 11: Annotation process using VATIC

Figure 11 shows how VATIC creates annotations on video. First, a user annotates objects in two frames. Then, VATIC creates automatic annotations of the same objects on the frames between the two user-designated frames.

1.2 Linear Interpolation

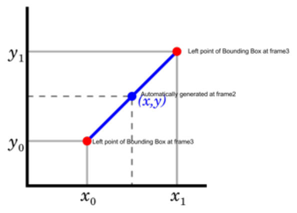

y_1
y_0
x_0
x_1
Left point of Bounding Box at frame3
Automatically generated at frame2
(x,y)
Left point of Bounding Box at frame3

Figure 12: Linear interpolation [8]

VATIC performs linear interpolation. Figure 12 shows, linear interpolation process of VATIC. For an example, a user sets a point in frame 1 and frame 3, then VATIC automatically generates a point in frame 2. If objects move nonlinearly, then VATIC does a poor job and a user needs to manually adjust the size and position of an annotation between two frames.

1.3 Tracking Integration with OpenCV

The main problem of VATIC is linear interpolation for creating annotation on video. Linear interpolation is inadequate for tracking non-linear behaviors of automobiles. A technical solution for the linear interpolation of VATIC is to implement tracking integration with OpenCV [9]. This is already implemented by John Doherty and it will show significant improvements in speed for the annotation of a video.

Chapter 5

Annotation

1.1 Annotation Methodology

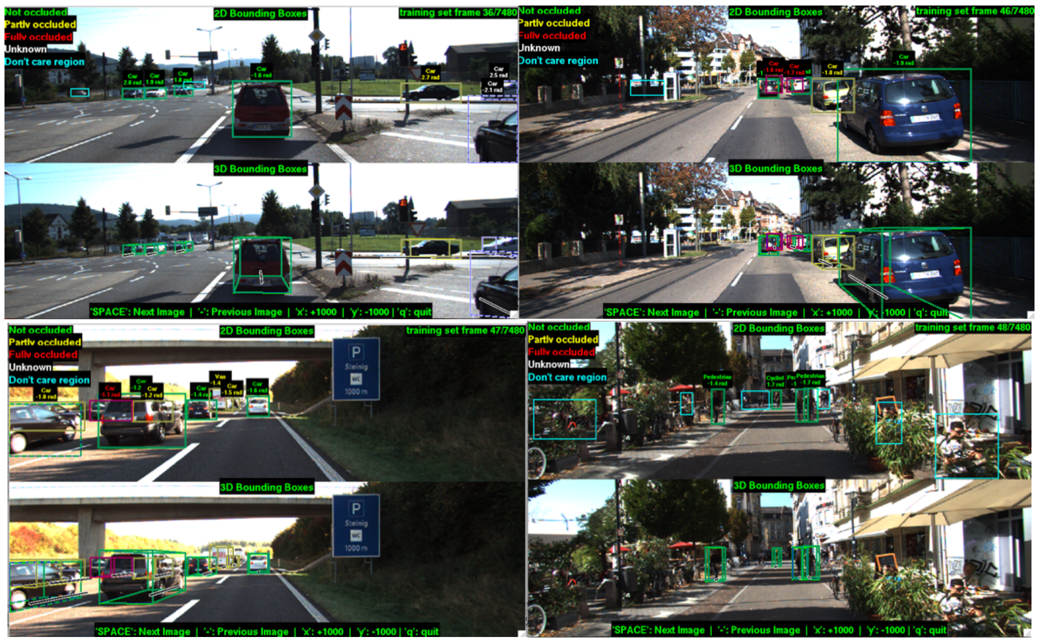


Figure 12: Annotation examples in KITTI dataset

We used the same annotation rules as KITTI. We annotated the following classes on videos: ‘cyclist’, ‘van’, ‘tram’, ‘car’, ‘misc’, ‘pedestrian’, ‘truck’, ‘person_sitting’, and ‘dontcare’.

After we observed annotations in the KITTI dataset, we realized that KITTI annotates a person as “pedestrian” if the person is standing and visible on the images, regardless of where they are standing. If a person is visible and riding something that is neither a bicycle nor a motorcycle, we annotated that person as “misc.” It is ambiguous because KITTI does not have such an image and we might need to think about different rules. Some objects, such as deer in Figure 2, we annotated as “misc.”

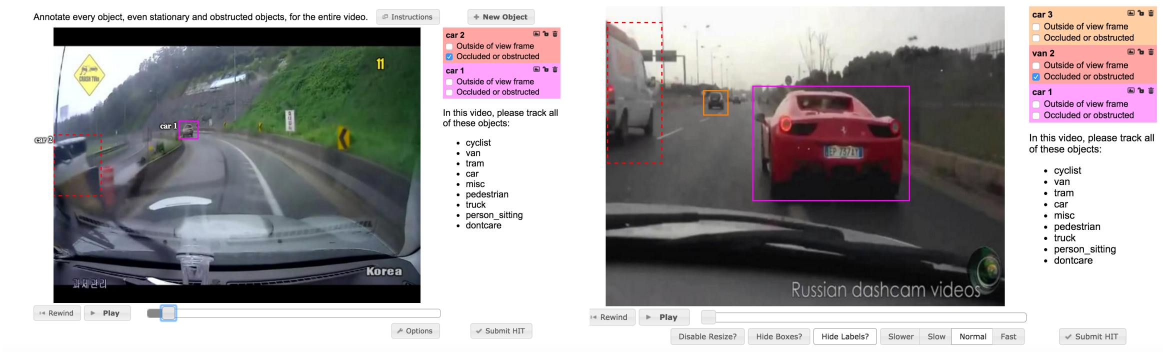
KITTI defines difficulties of its dataset for object detection as *easy, moderate,* or *hard* as follows [1]:
Easy: Minimum bounding box height: 40 Pixels,
Max. occlusion level: Fully visible.
Max. truncation: 15%.
Moderate: Min. bounding box height: 25 Pixels.
Max. occlusion level: Partly occluded.
Max. truncation: 30%.

Hard: Min. bounding box height: 25 Pixels
Max. occlusion level: Difficult to see.
Max. truncation: 50 %."

Objects in the KITTI dataset that have a minimum bounding box height less than 25 pixels are labeled as "dontcare." After finishing annotation using VATIC, we realized that it is not easy to annotate these "dontcare" instances. Also, if they are annotated by different people, then these "dontcare" objects might not really be reliable. It would be much more reliable and faster to annotate every instance and then change instances' classes in the annotation file by using scripts. Also, if there are too many cars or objects, then there is a problem. As a result we are limited to up to a maximum of 15 cars and 30 pedestrians per image as in the KITTI dataset.

To decide the most efficient workflow for Amazon Mechanical Turk, we timed how long annotation takes. Because VATIC supports only linear interpolation, it took more time. During annotation, we realized that often a camera shakes a lot during recording and linear interpolation does not help at all for these cases.

For a more effective process, integration of tracking using OpenCV [7] can be used. Another problem that appeared when using VATIC is labeling objects that are occluded. It was hard to label objects when they were occluded because sometimes they start out occluded but then become fully visible later. Lastly, after a worker annotates instances in VATIC, somebody should check his or her work because sometimes instances are added and not labeled.


Annotate every object, even stationary and obstructed objects, for the entire video.

car 2
- [ ] Outside of view frame
- [✓] Occluded or obstructed

car 1
- [ ] Outside of view frame
- [ ] Occluded or obstructed

In this video, please track all of these objects:
- cyclist
- van
- tram
- car
- misc
- pedestrian
- truck
- person_sitting
- dontcare

car 3
- [ ] Outside of view frame
- [ ] Occluded or obstructed

van 2
- [ ] Outside of view frame
- [✓] Occluded or obstructed

car 1
- [ ] Outside of view frame
- [ ] Occluded or obstructed

In this video, please track all of these objects:
- cyclist
- van
- tram
- car
- misc
- pedestrian
- truck
- person_sitting
- dontcare

Figure 13: Annotation examples in our dataset

Using VATIC, on average annotating every instance in a clip (10~13 seconds, 300 frames) takes about 23 minutes. Annotating one object in a video takes about 2 minutes, and a video clip generates about 300 images, which means we can annotate 300 instances in 2 minutes.

1.2 Results

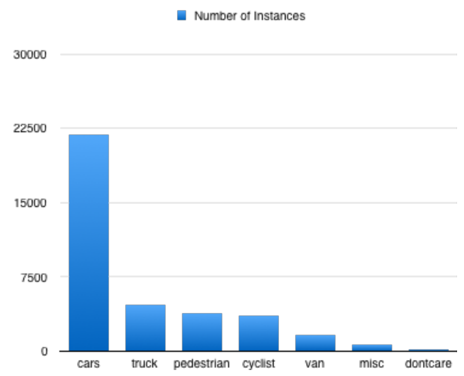


Figure 14: Distribution of the number of instances within an image

We were able to get 36,515 instances from 55 clips of video, which is 23,297 images. We have 21,852 cars, 4,756 trucks, 3,808 pedestrians, 3,613 cyclists, 1,684 vans, 651 misc., and 151 dontcares.

Once annotation was done, a development package is needed for training on any machine-learning framework. A framework such as Caffe [11] uses the PASCAL format and VATIC allows a user to create PASCAL-formatted data.

Chapter 7
Conclusion

The purpose of this research was to investigate some of the key problems that must be solved to create a dataset consisting of images that simulate driving conditions. The KITTI dataset is currently used for this purpose but it is small in size and uniform with regard to visual elements such as time of day and camera orientation. We aimed to investigate the feasibility of creating a dataset with less bias and more diverse driving situations.

Creating our dataset required us to consider many challenges that are common to creating a dataset for deep learning. First, once videos are collected from sources such as YouTube, many of them are duplicated. It was not possible to manually identify and remove all the duplicated video segments, even with multiple people. We resolved this issue by using a fingerprint method. We used images as the fingerprints of a file and found similarities among the videos. Bottlenecks due to serial work during the deduplication process was another technical problem we faced in constructing our dataset. As shown in Figure 8, instead of using a serial process, we divided videos into different directories and applied the deduplication in parallel. Using this process, we greatly reduced the number of hash-value comparisons for each image.

To be useful for supervised deep learning, the data needs to be annotated. There are many ways to create annotations for images, but we employed a tool called VATIC. VATIC provides faster annotation methods for video by supporting linear interpolation between frames. However, it takes a lot of extra effort to label every instance in images. One of the contributions of this project was to get a good average estimate of annotation time for this dataset. Using VATIC, on average, annotating every instance in a clip (10~13 seconds, 300 frames) takes about 23 minutes. Annotating one object in a video takes about 2 minutes, and a video clip generates about 300 images, which means we can annotate 300 instances in 2 minutes.

The results of this project enable subsequent researchers to estimate the effort to implement the amount of raw data, human effort, and computing resources needed to create a dataset for computer vision problems associated with self-driving cars. We also presented a solution to one of the key technical problems to be addressed, the elimination of duplicated data.

References

[1] Geiger, Andreas, Philip Lenz, and Raquel Urtasun. "Are we ready for autonomous driving? the kitti vision benchmark suite." In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 3354-3361. IEEE, 2012.
Also,
KITTI Object Detection Evaluation 2012 Dataset,
`http://www.cvlibs.net/datasets/kitti/eval_object.php`

[2] Krippendorff, Klaus. "Reliability in content analysis." *Human Communication Research* 30, no. 3 (2004): 411-433.

[3]Geiger, Andreas, Philip Lenz, Christoph Stiller, and Raquel Urtasun. "Vision meets robotics: The KITTI dataset." *The International Journal of Robotics Research* (2013): 0278364913491297.

[4] Brandon Castellano, `https://github.com/Breakthrough/PySceneDetect`

[5] Vondrick, Carl, Donald Patterson, and Deva Ramanan. "Efficiently scaling up crowdsourced video annotation." International Journal of Computer Vision 101, no. 1 (2013): 184-204.
Also, `http://web.mit.edu/vondrick/vatic/`

[6] Torralba, Antonio, and Alexei A. Efros. "Unbiased look at dataset bias." In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pp. 1521-1528. IEEE, 2011.

[7] John Doherty `https://github.com/johndoherty/vatic`

[8] Linear Interpolation,
`https://en.wikipedia.org/wiki/Linear_interpolation`

[9] John Doherty `https://github.com/gautamMalu/vatic_tracking`

[10] Keutzer, Kurt, and Tim Mattson. "A design pattern language for engineering (parallel) software." Intel Technology Journal 13, no. 4 (2010).

[11] Jia, Yangqing, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. "Caffe: Convolutional architecture for fast feature embedding." In *Proceedings of the ACM International Conference on Multimedia*, pp. 675-678. ACM, 2014.

[12] Ren, Shaoqing, Kaiming He, Ross Girshick, and Jian Sun. "Faster R-CNN: Towards real-time object detection with region proposal networks." In *Advances in Neural Information Processing Systems*, pp. 91-99. 2015.