SATYAM: DEMOCRATIZING GROUNDTRUTH FOR MACHINE VISION

Hang Qiu*, Krishna Chintalapudi†, Ramesh Govindan*
*University of Southern California †Microsoft Research

ABSTRACT

The democratization of machine learning (ML) has led to ML-based machine vision systems for autonomous driving, traffic monitoring, and video surveillance. However, true democratization cannot be achieved without greatly simplifying the process of collecting groundtruth for training and testing these systems. This groundtruth collection is necessary to ensure good performance under varying conditions. In this paper, we present the design and evaluation of **Satyam**, a first-of-its-kind system that enables a layperson to launch groundtruth collection tasks for machine vision with minimal effort. Satyam leverages a crowdtasking platform, Amazon Mechanical Turk, and automates several challenging aspects of groundtruth collection: creating and launching of custom web-UI tasks for obtaining the desired groundtruth, controlling result quality in the face of spammers and untrained workers, adapting prices to match task complexity, filtering spammers and workers with poor performance, and processing worker payments. We validate Satyam using several popular benchmark vision data sets, and demonstrate that groundtruth obtained by Satyam is comparable to that obtained from trained experts and provides matching ML performance when used for training.

1 INTRODUCTION

The accuracy of deep neural network based machine vision systems depends on the groundtruth data used to train them. Practically deployed systems rely heavily on being trained and tested on groundtruth data from images/videos obtained from actual deployments. Often, practitioners start with a model trained on public data sets and then fine-tune the model by re-training the last few layers using groundtruth data on images/videos from the actual deployment (Donahue et al., 2013; Retraining) in order to improve accuracy in the field.

Obtaining groundtruth data, however, can present a significant barrier, as annotating images/videos often requires significant human labor and expertise. Today, practitioners use three different approaches to groundtruth collection. Large companies employ their own trained workforce to annotate groundtruth. Third parties such as (Spare5) and (Figure Eight) recruit, train and make available a trained workforce for annotation. Finally, *crowdtasking platforms* such as Amazon Mechanical Turk (AMT (AMT, 2018)) provide rapid access to a pool of untrained *workers* that can be leveraged to generate groundtruth annotations.

While the first two approaches have the advantage of generating high quality groundtruth by using a trained workforce, they incur significant cost in recruitment and training, and are therefore often limited to well-funded companies. Consequently, employing a crowdtasking platform like AMT is often a preferred alternative for a large number of ML practitioners. Using AMT for obtaining groundtruth, however, presents several challenges that deter its widespread use. First, requesters may not always have the expertise needed to create user-friendly web user-interfaces to present to workers for annotation tasks. Second, worker quality varies widely in AMT, and results can be corrupted by spammers and bots, so requesters must curate results manually to obtain good groundtruth. Third, machine vision often requires groundtruth for hundreds or thousands of images and videos, and generate AMT Human Intelligence Tasks (HITs) manually is intractable, as is determining which workers need to be paid, or which workers to recruit.

Goal, Approach and Contributions. In this paper, we ask: *Is it possible to design a groundtruth collection system that is accessible to non-experts while also being both cost-effective and accurate?* To this end, we discuss the design, implementation, and evaluation of Satyam[1] (§2) which allows non-expert users to launch large groundtruth collection tasks in under a minute.

Satyam users first place images/video clips at a cloud storage location. They then specify groundtruth collection declaratively using a web-portal. After a few hours to a few days (depending on the size and nature of the job), Satyam generates the groundtruth in a consumable format and notifies the user. Behind the scenes, in order to avoid the challenges of recruiting and managing trained annotators, Satyam leverages AMT workers, but automates generation of customized web-UIs, quality control and HIT management.

High-level Specification. A key challenge in using AMT arises because the HIT is too low-level of an abstraction

[1]The Satyam portal (Satyam Portal) is functional, but has not yet been released for public use.

Figure 1: Examples of Satyam Results on Detection, Segmentation, and Tracking.

for large-scale groundtruth collection. Satyam elevates the abstraction for groundtruth collection by observing that machine vision tasks naturally fall into a small number of categories (§3), *e.g.,* classification (labeling objects in an image or a video), detection (identifying objects by drawing bounding boxes), segmentation (marking pixels corresponding to areas of interest) and a few others described in §3. Satyam allows users to specify their groundtruth requirements by providing customizable specification templates for each of these tasks.

Automated Quality Control. Satyam automates quality control in the face of an untrained workforce and eliminates the need for manual curation (§4). It requests annotations from multiple workers for each image/video clip. Based on the assumption that different workers make independent errors in the groundtruth annotation, Satyam employs novel *groundtruth-fusion* techniques that identify and piece together the "correct parts" of the annotations from each worker, while rejecting the incorrect ones and requesting additional annotations until the fused result is satisfactory.

Automated HIT Management - Pricing, Creation, Payment and Worker Filtering. Satyam automates posting HITs in AMT for each image/video in the specified storage location until the groundtruth for that image/video has been obtained. Instrumentation in Satyam's annotation web-UIs allow it to measure the time taken for each HIT. Satyam uses this information to adaptively adjust the price to be paid for various hits and ensures that it matches the requester's user-defined hourly wage rate. Satyam determines whether or not a worker deserves payment by comparing their work against the final generated groundtruth and disburses payments to deserving workers. When recruiting workers, it uses past performance to filter out under-performing workers.

Implementation and Deployment. Satyam's implementation is architected using a collection of cloud functions that can be auto-scaled, that support asynchronous result handling with humans in the loop, and that can be evolved and extended easily. Using an implementation of Satyam on Azure, we evaluate (§6) various aspects of Satyam. We find that Satyam's groundtruth almost perfectly matches groundtruth from well known publicly available image/video data sets such as KITTI (Geiger et al., 2012) which were annotated by experts or by using sophisticated equipment. Further, ML models re-trained using Satyam groundtruth perform identically with the same models re-trained with these benchmark datasets. We have used Satyam for over a year launching over 162,000 HITs on AMT to over 12,000 unique workers.

Examples of groundtruth generated by Satyam. Figure 1 show examples of the groundtruth generated by Satyam for detection, segmentation, and tracking, and how these compare with groundtruth from benchmark datasets. More examples are available in Figures 14, 15 and 16 and at (Tracking; Detection).

2 Satyam Overview

Satyam is designed to minimize friction for non-expert users when collecting groundtruth for ML-based machine vision systems. It uses AMT but eliminates the need for users to develop complex Web-UIs or manually intervene in quality control or HIT management.

2.1 Design Goals

We now briefly describe the key design goals that shaped the architectural design of Satyam.

Ease of use. Satyam's primary design goal is ***ease of use***. Satyam users should only be required to provide a high-level declarative specification of ground-truth collection: what kind of ground-truth is needed (*e.g.,* class labels, bounding boxes), for which set of images/videos, how much the user is willing to pay, *etc.* Satyam should not require user intervention in designing UIs, assessing result quality, or ensuring the appropriate HIT price, *etc.*, but must perform these automatically.

Scalability. Satyam will be used by multiple concurrent users, each running several different ground-truth collection activities. Each activity in turn might spawn several tens of thousands of requests to workers, and each request might involve generating web user interfaces, assessing results, and spawning additional requests, all of which might involve significant compute and storage.

Asynchronous Operation. Because Satyam relies on humans in the loop, its design needs to be able to ***tolerate large and unpredictable delays***. Workers may complete some HITs within a few seconds of launch, or may take days to process HITs.

Evolvability. In designing algorithms for automating ground truth collection, Satyam needs to take several design dimensions into account: the requirements of the user, the constraints of the underlying AMT platform, variability in worker capabilities, and the complexity of assessing visual annotation quality. These algorithms are complex, and will

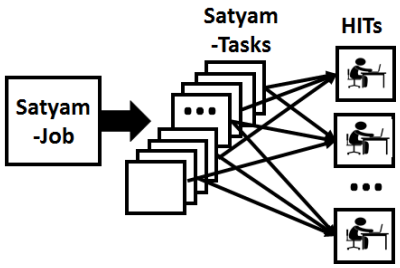

Figure 2: Satyam's jobs, tasks and HITs

evolve over time, and Satyam's design must accommodate this evolution.

Extensibility. ML for machine vision is rapidly evolving and future users will need new kinds of groundtruth data which Satyam must be able to accommodate.

2.2 Satyam Abstractions

Satyam achieves ease of use and asynchronous operation by introducing different abstractions to represent logical units of work in groundtruth annotation (depicted in Figure 2).

Satyam-job. Users specify their groundtruth collection requirements at a high-level as a ***Satyam-job***, which has several parameters: the set of images/video clips, the kind of groundtruth desired (*e.g.,* bounding rectangles for cars), payment rate ($/image or $/hour), the AMT requester account information to manage HITs on the user's behalf, *etc.* At any instant, Satyam might be running multiple Satyam-jobs.

Satyam-tasks. Satyam renders jobs to ***Satyam-tasks***, which represent the smallest unit of groundtruth work sent to a worker. For example, a Satyam-task might consist of a single image in which a worker is asked to annotate all the bounding rectangles and their classes (§1), or a short video clip in which a worker is asked to track one or more objects. A single Satyam-job might spawn hundreds to several tens of thousands of Satyam-tasks.

HIT. A ***HIT*** is an AMT abstraction for the smallest unit of work for which a worker is paid. Satyam decouples HITs from Satyam-tasks, for two reasons. First, Satyam may batch multiple Satyam-tasks in a HIT. For example, it might show a worker 20 different images (each a different Satyam-task) and ask her to classify the images as a part of a single HIT. Batching increases the price per HIT thereby incentivizing workers more, and also increases worker throughput. Second, it allows a single Satyam-task to be associated with multiple HITs, one per worker: this permits Satyam to obtain groundtruth for the same image from multiple workers to ensure high quality results.

2.3 Satyam Architecture

Satyam is architected as a collection of components (Figure 4) each implemented as a *cloud function* (*e.g.,* an Azure function or an Amazon lambda) communicating through persistent storage. This design achieves several of Satyam's goals. Each component can be scaled and evolved independently. Components can be triggered by users requesting new jobs or workers completing HITs and can thereby accommodate asynchronous operation. Finally, only some components need to be modified in order to extend Satyam to new types of ground truth collection.

Satyam's components can be grouped into three high-level functional units, as shown in Figure 4: *Job Rendition*, *Quality Control* and *HIT Management*. We describe these components, and their functional units, in the subsequent sections.

Job Rendition. This functional unit raises the level of abstraction groundtruth collection (Section 3). It is responsible for translating the user's high level groundtruth collection requirements to AMT HITs and then compiling the AMT worker results into a presentable format for users. Users primarily interact with the *Job-Submissions Portal* (Satyam Portal) where they submit their groundtruth collection requirements. Submitted jobs are written to the *Job-Table*. Based on the job descriptions in the Job-Table, the *Preprocessor* may perform data manipulations such as splitting videos into smaller chunks. The *Task Generator* decomposes the Satyam-job into Satyam-tasks, one for each image/video chunk. The *Task Portal* is a web application that dynamically renders Satyam-tasks into web pages (based on their specifications) displayed to AMT workers. Finally, *Groundtruth Compilation* assembles the final results from the workers for the entire job and provides them to Satyam users as a JSON format file.

Quality Control. AMT workers are typically untrained in groundtruth collection tasks and Satyam has little or no direct control over them. Further, some of the workers might even be bots intending to commit fraud (mturk-spam). The quality control components are responsible for ensuring that the groundtruth generated by Satyam is of high quality. In order to achieve this, Satyam sends the same task to multiple non-colluding workers and combines their results. The *Result Aggregator* identifies and fuses the "accurate parts" of workers' results while rejecting the "inaccurate parts" using *groundtruth fusion algorithms* described in §4. For certain tasks, the Aggregator might determine that it requires more results to arrive at a conclusive high quality result. In that case, it presents the task to more workers until a high quality groundtruth is produced. The *Results Evaluator* compares fused results with the individual worker's results to determine whether the worker performed acceptably or not and indicates this in the *Result-Table*.

HIT-Management. These components directly interact with the Amazon AMT platform and manage HITs through their life-cycle (§5). The *HIT Generator* reads the task table and launches HITs in AMT and always ensures that there

Job Category	Description	Coverage
Image Classification	Select class name of displayed image	22.5%
Video Classification	Select class name of displayed video	5.3%
Object Detection	Draw/edit bounding boxes and select class labels for each object of interest in an image	25.9%
Object Tracking	Draw/edit bounding boxes and select class labels for each object of interest in an image	10.9%
Object Segmentation	Draw arbitrary polygons around various areas of interest in an image	33.1%
Object Counting	Count the number of objects in an image or a video clip	0%
OCR	Recognize texts in an image	2.1%

Figure 3: Satyam Job Categories

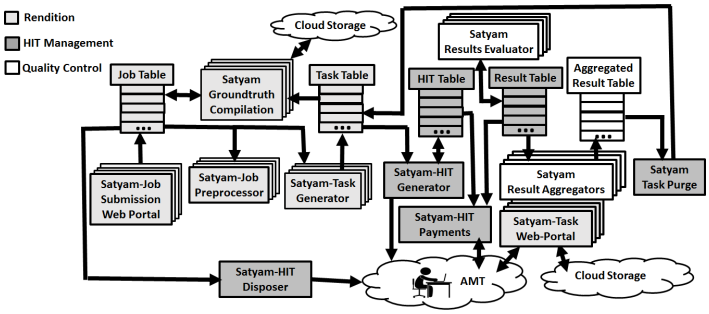

Figure 4: Overview of the Satyam's components

are no unfinished tasks with no HITs. It is also responsible for *adaptive pricing* – adaptively adjusting the HIT price by measuring the median time to task completion, and *worker filtering* – ensuring that under-performing workers are not recruited again. The *HIT Payments* component reads the results table and pays workers who have completed a task acceptably, while rejecting payments for those who have not. The *Task Purge* component removes tasks from the task table that have already been aggregated, so that they are not launched as HITs again and the *HIT Disposer* removes any pending HITs for a completed job.

3 Job Rendition

To achieve ease of use, Satyam needs to provide users with an expressive high-level specification framework for ground truth collection. Satyam leverages the observation that, in the past few years, the machine vision community has organized its efforts around a few well-defined categories of *vision tasks*: classification, detection or localization, tracking, segmentation, and so forth. Satyam's job specification is based on the observation that different ground truth collection within the same vision task category (*e.g.,* classification of vehicles vs. classification of animals) share significant commonality, while ground-truth collection for different vision task categories (*e.g.,* classifying vehicles vs. tracking vehicles) are qualitatively different.

Job Categories. Satyam defines a small number of *job categories* where each category has similar ground-truth collection requirements. Users can customize groundtruth collection by parameterizing a job category *template*. For example, to collect class label groundtruth for vehicles (*e.g.,* car, truck, *etc.*), a user would select an image classification job template and specify the various vehicle class labels.

Templatizing job categories also enables Satyam to automate all steps of ground-truth collection. The web UIs presented to AMT workers for different ground truth collection jobs in the same category (*e.g.,* classification) are similar, so Satyam can automatically generate these from *Web-UI templates*. Moreover, quality control algorithms for ground truth collection in the same category are similar (modulo simple parametrization), so Satyam can also automate these.

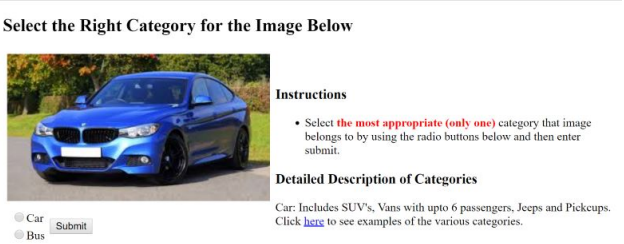

Figure 5: Image Classification Task Page

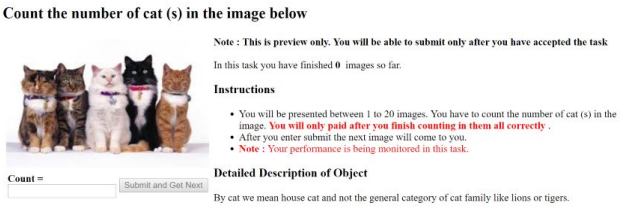

Figure 6: Object Counting Task Page

To determine which job categories to support, we examined the top 400 publicly available groundtruth datasets used by machine vision researchers (YACVID) and categorized them with respect to the Web-UI requirements for obtaining the groundtruth (Figure 3). The *coverage* column indicates the fraction of datasets falling into each category. Satyam currently supports the first six categories in Figure 3, which together account for the groundtruth requirements of more than 98.1% of popular datasets in machine vision. We now briefly describe a few of the most used currently available templates in Satyam.

Image and Video Classification. The desired groundtruth in this category is the label (or labels), from among a list of provided class labels, that most appropriately describes the image/video. Class labels can describe objects in images such as cars or pedestrians and actions in video clips such as walking, running, and dancing. Satyam users customize (Figure 5) the corresponding job templates by providing the list of class labels and a link or description for them. To the workers, the web-UI displays the image/video clip with the appropriate instructions and a radio button list of class labels.

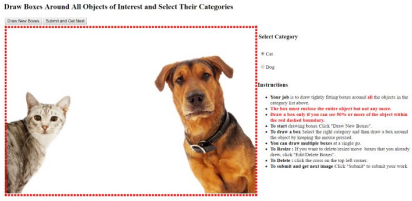

Figure 7: Object Detection and Localization Task Page

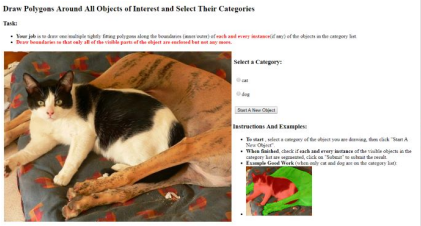

Figure 8: Object Segmentation Task Page

Figure 9: Multi-Object Tracking Task Page

Object Counting in Images and Videos. The desired groundtruth for this job category is a count of objects of a certain class, or of events in an image or video (*e.g.,* the number of cars in a parking lot or the number of people entering a certain mall or airport). The user provides a description of the object/event. In the web-UI, the worker is shown an image/video clip (Figure 6), and the description provided of the object/event of interest, for which the worker is asked to provide a count.

Object Detection in Images. The desired groundtruth in this category is a set of bounding boxes on an image marking parts of interest in the image, along with a class label that most appropriately describes each box. Users specify (Figure 7) the object classes for which workers should draw bounding boxes describing areas within each image that need to be annotated. Workers see an image with a radio button list of object classes, using which the workers can select one class to draw/edit bounding boxes around all objects of the same class, *e.g.,* all pedestrians in a traffic surveillance image, in one shot.

For example, in a traffic surveillance scene, the objects of interest might be all the cars and pedestrians. The groundtruth required for such algorithms for each image, is the set of all bounding boxes enclosing the objects of interest and their respective category names. To support these cases we provide a template that generates a web-UI where workers are displayed an image and can draw/edit bounding boxes around objects of interest (using the mouse). A radio button list of the categories helps the workers categorize the object as well. Satyam users customize this template by specifying the categories of interest. The users may also specify a set of polygons describing the various areas of interest within the images.

Object Segmentation in Images. The desired groundtruth in this category is pixel-level annotations of various objects/areas of interest (*e.g.,* people, cars, the sky). This template is similar to the object detection template except that it lets workers annotate arbitrary shapes by drawing a set of polygons (Figure 8).

Object Tracking in Videos. The desired groundtruth in this category, an extension of object detection to videos, requires bounding boxes for each distinct object/event of interest in successive frames of a video clip. This groundtruth can be used to train object trackers. Satyam users can select (Figure 9) the video tracking job category, and specify the object classes that need to be tracked, instructions to workers on how to track them, what frame rate the video should be annotated at, and polygons that delineate areas of interest within frames. Workers are presented (Figure 9) with a short video sequence, together with the categories of interest, and can annotate bounding boxes for each object on each frame of the video. For annotation, we have modified an existing open source video annotation tool (Vatic) and integrated it into Satyam.

Job Rendition Components. When a user wishes to initiate ground truth collection, she uses the *Job Submission Portal* to select a job category template, and fills in the parameters required for that template. Beyond the category specific parameters described above, users provide a cloud storage location containing the images or videos to be annotated, and indicate the price they are willing to pay. After the user submits the job specification, the Portal generates a globally unique ID (GUID) for the job, and stores the job description in the *Job-Table*. Then, the following components perform job rendition.

Pre-processor. After a job is submitted via the Job Submission portal, the images/video clips might need to be preprocessed. In our current implementation, Satyam supports preprocessing for video annotations. Specifically, large videos (greater than 3 second duration) are broken into smaller chunks (with a small overlap between successive chunks to facilitate reconstruction or *stitching*, see below) to diminish cognitive load on workers. They are then downsampled based on user's requirements, and converted into a browser-friendly format (*e.g.,* MP4).

Task Generator. This component creates a Satyam-task for each image or video chunk. A Satyam-task encapsulates all the necessary information (image/video URI, user customizations, the associated Job-GUID, *etc.*) required to render a web-page for the image/video clip. The Satyam-task is stored as a JSON string in the *Task-Table*. The Task Table stores additional information regarding the task, such as the number of workers who have attempted it.

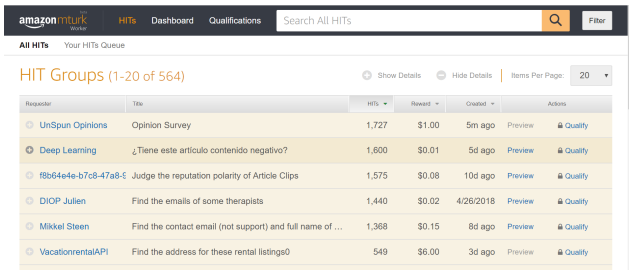

Figure 10: Amazon MTurk HITs Web Portal

Task Web-UI Portal. An AMT worker sees HITs listed by the title of the template and the price promised for completing the HIT (Figure 10). (At any given instant, Satyam can be running multiple Satyam-jobs for each supported template). When the worker *accepts* a HIT, she is directed to the Satyam Task Web-UI Portal, which dynamically generates a web page containing one or more Satyam-tasks. For example, Figure 9 shows a Web-UI page for the tracking templates. The generated web page appears as an IFrame within the AMT website. When the worker submits the HIT, the results are entered into the *Result-Table* and AMT is notified of the HIT completion.

When dynamically generating the web page, Satyam needs to determine which Satyam-tasks to present to the worker. Listing HITs only by task portal and by price allows *delayed binding* of a worker to Satyam-tasks. Satyam uses this flexibility to (a) achieve uniform progress on Satyam-tasks and (b) avoid issuing the same task to the same worker. When a worker picks a HIT for template T and price p, Satyam selects that Satyam-task with the same T and p which has been worked upon the least (using a random choice to break ties). There is on exception to this *least-worked-on* approach. Satyam may need to selectively finish aggregating a few tasks to gather statistics for dynamic price adjustment (described later). In such instances, the least-worked-on mechanism and randomization is restricted to a smaller subgroup rather than the whole task pool, so that the subgroup completes quickly. To avoid issuing the same task to the same worker, Satyam can determine, from the task table, if the worker has already worked on this task (it may present the same task to multiple workers to improve result quality, §4). A single HIT may contain multiple Satyam-tasks, so Satyam repeats this procedure until enough tasks have been assigned to the HIT.

Groundtruth Compilation. Once all the tasks corresponding to a job have been purged (§5), this component compiles all the aggregated results corresponding to this job into a JSON file, stores that file at a user-specified location and notifies the user. Before ground-truth compilation, Satyam may need to *post-process* the results. Specifically, for video-based job categories like tracking, Satyam must *stitch* video chunks together to get one seamless groundtruth for the video. We omit the details of the stitching algorithm, but it uses the same techniques as the groundtruth-fusion tracking algorithm (§4.2) to associate elements in one chunk with those in overlapped frames in the next chunk.

4 Quality Control

Satyam's quality control relies on the wisdom of the crowds (Surowiecki, 2005): *when a large enough number of non-colluding workers independently agree on an observation, it must be "close" to the groundtruth*. To achieve this Satyam solicits groundtruth for the same image/video clip from multiple workers and only accepts elements of the groundtruth that have been *corroborated* by multiple workers. For instance, in a detection task with several bounding boxes, only those, for which at least 3 workers have drawn similar bounding boxes, are accepted.

Figure 11 depicts Satyam's quality control loop. One instance of the loop is applied to each Satyam-task. Satyam first sends the same task to n_{min} workers to obtain their results. n_{min} depends on the job category and is typically higher for more complex tasks (described in more detail below). In the *groundtruth-fusion* step, Satyam attempts to corroborate and fuse each groundtruth element (*e.g.,* bounding box) using a job category specific groundtruth-fusion algorithm. If the fraction of corroborated elements in an image/video clip is less than the coverage threshold (η_{cov}), Satyam determines that more results need to be solicited and relaunches more HITs, one at a time. For some images/videos, even for humans, agreeing on groundtruth maybe difficult. For such tasks we place a maximum limit n_{max} (20 in our current implementation) on the number workers we solicit groundtruth from. The task is marked "aggregated" and removed from the task list either if we reach the maximum limit or if the fraction of corroborated elements exceeds η_{cov}.

4.1 Dominant Compact Cluster

All the groundtruth-fusion algorithms in Satyam are based on finding the *Dominant Compact Cluster* (DCC) which represents *the set of similar results that the largest number of workers agree on*. If the number of elements in the dominant compact set is greater than n_{corr}, the groundtruth for that element is deemed as corroborated.

Definition. Suppose that n workers have generated n versions of the groundtruth $E_1, E_2, ..., E_n$ for a particular element in the image/video (as in Figure 12 where each of the 4 workers has drawn bounding box around the orange car). For each job category, we define a distance metric $D(E_i, E_j)$ that is higher the more dissimilar E_i and E_j are. A fusion function $F_{fusion}(E_1, E_2, \cdots, E_k) = E_{fused}$ specifies how different versions of the groundtruths can be combined into one (*e.g.,* by averaging multiple bounding boxes into one). All *groundtruth-fusion* algorithms start by

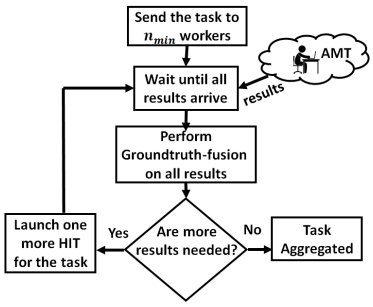

Figure 11: Quality Control Loop in Satyam

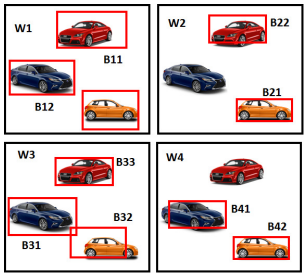

Figure 12: Example groundtruth fusion in Multi-object Detection

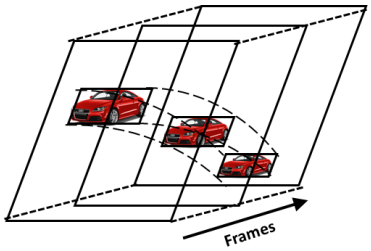

Figure 13: Groundtruth fusion in Multi-object Tracking is a 3-D extension of Multi-object detection

clustering $E_1, E_2, \cdots, E_n$ based on D while guaranteeing that none of the elements of its cluster is farther than τ distance from the fused element *i.e.,* $D(E_{fused}, E_k) < \tau$ for all E_k within a cluster. τ, the *compactness constraint*, ensures that the clusters do not have any results that are too dissimilar from each other. After the clustering, the cluster with the most number of elements is deemed the *dominant compact cluster* and E_{fused} computed over this cluster is deemed the cluster head.

Greedy Hierarchical Clustering to find DCC. Finding DCC is NP-Hard, so we use greedy hierarchical clustering. We start with n clusters, the i^{th} cluster having the one element E_i. At each step, two clusters with the closest cluster heads are merged, provided that the merged cluster does not violate the compactness constraint. The clustering stops as soon as clusters cannot be merged any longer.

Variations across different templates. While finding the DCC is common across all groundtruth-fusion algorithms, the specific values and functions such as n_{min}, $D(E_i, E_j)$, n_{corr}, F_{fusion}, η_{cov} and τ are different for each fusion algorithm. In the rest of this section, we describe the various choices we use for these values and functions.

4.2 Fusion Details

Image and Video Classification. For this template, Satyam uses a super-majority criterion, selecting that class for which the fraction of workers that agree on the class exceeds $\beta \in (0, 1)$ (we chose $\beta = 0.7$, §6.8). This is equivalent to the DCC algorithm with the distance function $D = 0$ if two workers choose the same category and ∞ if they do not, $\tau = 0$, and $n_{corr} = \beta n$, where n is the number of results.

Counting in Images/Videos. Given n counts, by n workers, our goal is to robustly remove all the outliers and arrive at a reliable count. We use DCC for this, with $D(C_i, C_j) = |C_i - C_j|$ where C_i and C_j are the counts from the i^{th} and j^{th} workers. F_{fusion} is chosen as the average of all the counts. $\tau = \lfloor(\epsilon C)\rfloor$, where C is the average count of the cluster, *i.e.,* two counts are deemed to be similar only if their deviation is less than $\pm\epsilon$ fraction of the average count. We chose $\epsilon = 0.1$ in our implementation. n_{min} and n_{max} are chosen to be 10 and 20 respectively (§6.8).

Object Detection in Images. To provide intuition into the groundtruth fusion algorithm for this template we use the example in Figure 12, where four workers have drawn bounding boxes around cars in an image. The j^{th} bounding box drawn by the i^{th} worker is represented by B_{ij}. A worker may not draw bounding boxes for all cars (*e.g.,* W_2 and W_4), and two different workers may draw bounding boxes on the same image in a different order (*e.g.,* W_1 draws a box around the red car first, but W_3 does it last). Furthermore, workers may not draw bounding boxes consistently: W_3's box around the orange car is off-center and box B_{11} is not tightly drawn around the red car. Our fusion algorithm, designed to be robust to these variations.

Bounding Box Association. Since different workers might draw boxes in a different order, we first find the correspondence between the boxes drawn by the different workers. In Figure 12, this corresponds to grouping the boxes in the three sets $G_1 = \{B_{11}, B_{22}, B_{33}\}$, $G_2 = \{B_{12}, B_{31}, B_{41}\}$ and, $G_3 = \{B_{13}, B_{21}, B_{32}, B_{42}\}$ where each set has boxes belonging to the same car. We model this problem as a multipartite matching problem where each partition corresponds to the bounding boxes of a worker, and the goal is to match bounding boxes of each worker for the same car.

To determine the matching, we use a similarity metric, *Intersection over Union (IoU)*, between two bounding boxes, which is the ratio of intersection of the two bounding boxes to their union. Since the matching problem is NP-Hard, we use an iterative greedy approach. For a total of N bounding boxes, we start with N sets with one bounding box per set. At each iteration, we merge the two sets with the highest average similarity while ensuring that a set may have only one bounding box from a partition. The algorithm terminates when there are no more sets that can be merged. In the end, each set corresponds to all the boxes drawn by different workers for one distinct object in the image.

Applying groundtruth-fusion on each object. Once we know the set of bounding boxes that correspond to each other, we can use DCC for fusion. Let bounding box $B_i = \langle x_i^{tl}, y_i^{tl}, x_i^{br}, y_i^{br} \rangle$ where (x_i^{tl}, y_i^{tl}) and (x_i^{br}, y_i^{br}) are the top left and bottom right pixel coordinates respectively. We choose $D(B_i, B_j) = \max(|x_i^{tl} - x_j^{tl}|, |y_i^{tl} - y_j^{tl}|, |x_i^{br} - x_j^{br}|, |y_i^{br} - y_j^{br}|)$, $n_{corr} = 3$, $\tau = 15$ (pixels), $n_{cov} = 0.9$, $n_{min} = 5$, $n_{max} = 20$. The fusion-function F_{fusion} generates a fused bounding box as the average of each of top-left and bottom-right pixel coordinates of all the bounding boxes being fused. Thus, in order to be similar, none of the corners of the boundaries must deviate by more than τ pixels along the x or y axis. The minimum number of workers to corroborate each box is 3 and 90% of the boxes need to corroborated before the quality control loop terminates. We arrived at these parameters through a sensitivity analysis (§6.8).

Object Segmentation in Images. The fusion algorithm used for image segmentation is almost identical to that used for multi-object detection except that bounding boxes are replaced by segments: arbitrary collections of pixels. Thus, while associating segments instead of bounding boxes, the IoU metric is computed by considering individual pixels common to the two segments. For F_{fusion}, a pixel is included in the fused segment only if it was included in the annotations of at least 3 different workers. We use $\tau = 1/0.3$, $n_{corr} = 3$, $n_{min} = 10$, $n_{max} = 20$ and $\eta_{cov} = 0.9$.

Object Tracking in Videos. Fusion algorithm for multi-object tracking simply extends that used for multi-object detection to determine a fused *bounding volume*, a 3-D extension of bounding box (as shown in Figure 13). We extend the definition of IoU to a bounding volume by computing and summing intersections and unions over each frame, deemed *3D-IoU*. For F_{fusion}, we average the bounding boxes across users at each frame independently; this is because different workers may start and end the track at different frames. We use $\tau = 1/0.3$, $n_{corr} = 3$, $n_{min} = 5$, $n_{max} = 20$ and $\eta_{cov} = 0.9$.

4.3 Result Evaluation

After all the results for a task have been fused, Satyam *approves* and pays or *rejects* each worker's HIT (§5).

For image and video *classification*, Satyam approves all HITs in which the worker's selected class matches that of the aggregated result. When no class label achieves a super-majority (§4.2), it ranks all classes in descending order of the number of workers who selected them, then chooses the minimum number of categories such that the combined number of workers that selected them is a super-majority, and approves all their HITs. For *counting*, Satyam approves each worker whose counting error is within ϵ of the fused count (§4.2). For object *detection, segmentation and tracking*, Satyam approves each worker whose work has contributed to most of the objects in the image/video. Specifically, Satyam approves a worker if the bounding boxes generated by the worker were in more than half of the dominant compact clusters (§4.2) for objects in the image.

5 HIT MANAGEMENT

These components manage the interactions between Satyam and AMT such as launching HITs for the tasks, estimating and adapting the price of HITs to match user specifications, filtering under-performing workers, submitting results to the quality control component, and finally, making/rejecting payments for tasks that have completed.

HIT Generator. This component creates HITs in AMT using the web-service API that AMT provides (MTurk SDK) and associates these HITs with an entry in the *HIT-Table* (which also contains pricing metadata, as well as job/task identification). It ensures that every unfinished task in the Satyam-task table has at least one HIT associated with it in AMT. It does this by comparing the number of unfinished tasks in the Task-Table for each GUID and price level against the number of unfinished HITs in the HIT-Table and determines the deficit. Because a single HIT may comprise multiple tasks, Satyam computes the number of extra HITs needed to fill any deficit and launches them. To determine which HITs have been worked on, as soon as a worker submits a HIT, Satyam records this in the HIT-Table.

HIT Price Adaptation. Several organizational and state laws require hourly minimum wage payments. Moreover, hourly wages are easier for users to specify. However, payments in AMT are disbursed at the granularity of a HIT. Thus, Satyam must be able to estimate the "reasonable" time taken to do a HIT and translate it to price per HIT based on the desired hourly rate. The time taken for a HIT can vary from a few seconds to several minutes and depends on three factors: (a) the type of the template (*e.g.,* segmentation tasks take much longer than classification tasks); (b) even within the same template, more complex jobs can take longer *e.g.,* scenes with more cars at a busy intersection; (c) finally, different workers work at different rates.

To estimate HIT completion times, Satyam instruments the web-UIs provided to workers and measures the time taken by the worker on the HIT. As each job progresses, Satyam continuously estimates the median time to HIT completion per job (considering only approved HITs). It uses this value to adjust the price for each future HIT in this particular job. Using this, Satyam's price per HIT converges to conform to hourly minimum wage payments (§6).

Satyam HIT Payments. Once a task is aggregated, deserving workers must be paid. Satyam relies on the fusion algorithms to determine whether a result should be *accept*-ed or not (§4). A single HIT may include multiple Satyam-tasks;

Satyam's HIT Payments component computes the fraction of *accepted* results in a HIT across all of these tasks and pays the worker if this fraction is above a threshold.

Worker Filtering. Worker performance can vary across templates (*e.g.,* good at classification but not segmentation), and across jobs within a given template (*e.g.,* good for less complex scenes but not for more complex ones). To minimize rejected payments, Satyam tracks worker performance and avoids recruiting them for tasks they might perform poorly at. To do this, as Satyam rejects payments to undeserving workers for a certain task, it tracks worker approval rates (using the AMT-supplied opaque workerID) for each job and does not serve HITs to workers that have low approval rates (lower than 50% in our implementation). While serving HITs to workers with past high performance history allows Satyam to be efficient, Satyam must also explore and be able to discover new workers. Thus, Satyam allows workers with good approval rates to work on 80% of the HITs, reserving the rest for workers for whom it does not have any history. As shown in our evaluations 6.7, worker filtering results in much fewer overall rejections.

Satyam Task Purge. This component, triggered whenever a result is aggregated, removes completed tasks from the Task Table so that they no longer show up in any future HITs.

6 EVALUATION

We have implemented all components (Figure 4) of Satyam on Azure. Our implementation is 13635 lines of C# code. Using this, we evaluate Satyam by comparing the fidelity of its groundtruth against public *ML benchmark* datasets. In these benchmarks, groundtruth was curated/generated by trained experts or by using specialized equipment in controlled settings. To demonstrate Satyam's effectiveness in a real world deployment we generate a data set by extracting images from four video surveillance streams at major traffic intersections in two US cities.

We evaluate Satyam along the following dimensions: (a) The quality of ground truth obtained by Satyam compared with that available in popular benchmark data sets; (b) The *accuracy* of deep neural networks trained using groundtruth obtained by Satyam compared with those trained using benchmark data sets; (c) The efficacy of fine-tuning in a deployed real-world vision-based system; (d) The *cost* and *time* to obtain groundtruth data using Satyam and; (e) The efficacy of our adaptive pricing and worker filtering algorithms (f) The sensitivity of groundtruth-fusion algorithms to parameters.

6.1 ML Benchmark Datasets

Image Classification (ImageNet-10). We create this dataset by picking all the images corresponding to 10 classes commonly seen in video surveillance cameras from the ImageNet (Deng et al., 2009) dataset. Our dataset contains 12,482 images covering these classes: cat, dog, bicycle, lorry-truck, motorcycle, SUV, van, female person and male person.

Video Classification (JHMDB-10). For this data set we pick all the video clips from the JHMDB (Jhuang et al., 2013) data set corresponding to to 10 common human activities: clap, jump, pick, push, run, sit, stand, throw, walk and, wave (a total of 411 video clips).

Counting in Images (CAPRK-1). We create this data set by selecting 164 drone images taken from one parking lot from CAPRK (Hsieh et al., 2017) (a total of 3,616 cars).

Object Detection in Images (KITTI-Object). We create this data set by considering 3 out of 8 classes (cars, pedestrians and cyclists) in the KITTI (Geiger et al., 2012) data set with 8000 images (a total of 20,174 objects.). The groundtruth in KITTI established using LiDAR mounted on the car.

Object Segmentation in Images (PASCAL-VOC-Seg). PASCAL-VOC (Everingham et al., 2015) is a standardized image dataset for object classification, detection, segmentation, action classification, and person layout. We create this data set by choosing 353 images from the PASCAL-VOC (Everingham et al., 2015) data set that have segmentation labels, including the groundtruth of both class- and instance-level segmentation, corresponding to a total of 841 objects of 20 different classes.

Tracking in Videos (KITTI-Trac). For this dataset we chose all 21 video clips that were collected from a moving car from KITTI (Geiger et al., 2012) (about 8000 frames), but evaluate tracks only for 2 classes – cars and pedestrians. During the pre-processing step, these 21 video clips were broken into 276 chunks of length 3 seconds each with a 0.5 second overlap between consecutive chunks.

Traffic Surveillance Video Stream Data (SURV). We extracted images at 1 frame/minute from the video streams of 4 live HD quality traffic surveillance cameras, over one week (7 days) between 7:00 am and 7:00 pm each day. These cameras are located at major intersections in two U.S cities. We label the dataset corresponding to each of the four cameras as SURV-1, SURV-2, SURV-3 and SURV-4 respectively.

6.2 Quality of Satyam Groundtruth

To demonstrate that Satyam groundtruth is comparable to that in the ML benchmarks, we launched a job in Satyam

Figure 14: Example Results of Satyam Detection from KITTI

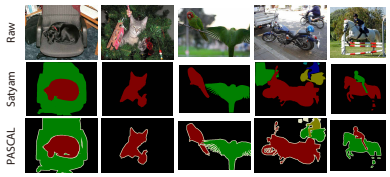

Figure 15: Example Results of Satyam Segmentation from PASCAL

Figure 16: Example Results of Satyam Tracking from KITTI

	Video Classification	Image Classification	Object Counting	Detection	Image Segmentation	Tracking
Dataset	**JHMDB-10**	**ImageNet-10**	**CARPK-1**	**KITTI-Obj**	**PASCAL-VOC-Seg**	**KITTI-Trac**
# Objects Annotated	411	12482	3616	20174	841	1845
Precision	**99.29%**	**98.56%**	**96.92%**	**99.01%**	**94.77%**	**94.61%**
Recall	**99.22%**	**99.16%**	**N/A**	**97.13%**	**94.65%**	**95.86%**
Median Latency [hrs]	**7.4**	**8.75**	**9.5**	**170.69**	**72.5**	**64.2**
Latency/Object [sec]	**64.82**	**2.5**	**9.46**	**30.46**	**310.34**	**125.27**
Avg. # Paid Results / Task	2.94	5.38	6.89	8.61	8.26	11.74
Median Time/Task [sec]	8.7	5.67	32.57	46.86	172.5	557
Mean # Objects/Task	1	1	22.04	2.52	2.38	6.68
Median Time/Object [sec]	8.7	5.67	1.48	18.6	72.48	83.38
Person-Seconds/Object	**25.58**	**30.5**	**10.18**	**160.11**	**598.68**	**978.92**

Figure 17: Satyam Accuracy, Latency, and Cost

for each of the six benchmark data sets described in Figure 17. Figures 14, 15 and 16 show some examples of groundtruth obtained using Satyam for detection, segmentation and tracking templates respectively. For this comparison, we evaluate *match-precision* (the degree to which Satyam's groundtruth matches that of the benchmark) and *match-recall* (the degree to which Satyam's workers identify groundtruth elements in the benchmark).

Figure 17 summarizes Satyam's accuracy for the various templates relative to the benchmarks. Satyam has uniformly high match-precision (95-99%) and high match-recall (>95%) for the relevant benchmarks. We find that Satyam often deviates from the benchmark because *there are fundamental limits achieving accuracy with respect to popular benchmark data sets*, for two reasons. First, some of the benchmarks were annotated/curated by human experts and have a small fraction of errors or ambiguous annotations themselves. Some of the ambiguity, especially in classification, arises from linguistic confusion between class labels (*e.g.,* distinguishing between van and truck). Second, in others that were generated using specialized equipment (*e.g.,* LiDAR), part of the generated groundtruth is not perceivable to human eye itself. In the rest of this section, we describe our methodology for each job category and elaborate on these fundamental limits.

Image Classification. Satyam groundtruth for ImageNet-10 has a match-precision of 98.5% and a match-recall of 99.1%. The confusion matrix (Figure 19) for all the 10 categories in ImageNet-10, shows that the largest source of mismatch is from 10% of vans in ImageNet being classified as lorry-trucks by Satyam. We found that all of vehicles categorized as vans in ImageNet are in fact food or delivery trucks (*e.g.,* Figure 18), indicating *linguistic confusion* on the part of workers. The only other significant off-diagonal entry in Figure 19 at 1.6% results from linguistic confusion

Figure 18: Linguistic confusion between van and truck

between Vans and SUVs. Discounting these two sources of error, Satyam matches 99.9% of the groundtruth.

Video Classification. Satyam's groundtruth for this category set has a match-precision and match recall exceeding 99%. The confusion matrix (Figure 20) for the 10 categories in JHMDB-10 reveals only 3 mismatches compared to the benchmark groundtruth. We examined each case, and found that the errors resulted from class label confusion or from incorrectly labeled groundtruth in the benchmark: a person picking up his shoes was labeled as *standing* instead of *picking*; a person moving fast to catch a taxi was labeled as *walking* instead of *running*; and finally a person who was picking up garbage bags and throwing them into a garbage truck was labeled as *picking up* in JHMDB-10, while Satyam's label was *throwing*. Discounting these cases, Satyam matches 100% with the groundtruth.

Counting in Images. Satyam's car counts deviate from the CAPRK-I benchmark's count groundtruths by 3% (Figure 17), which corresponds to an error of 1 car in a parking lot with 30 cars. This arises because of cars that are only partially visible in the image (*e.g.,* Figure 21), and workers were unsure whether to include these cars in the count or not. By inspecting the images we found that between 3 and 10% of the cars in each image were partially visible.

Object Detection in Images. For quantifying the accuracy for this template, we adopt the methodology recommended by the KiTTI benchmark – two bounding boxes are said to match if their IoU is higher than a threshold. Satyam has a high match-precision of 99% and match-recall of 97% (Figure 17). The match-recall is expected to be lower than match-precision: the LiDAR mounted on KITTI's data collection vehicle can sometimes detect objects that may not be visible to the human eye.

Object Segmentation in Images. We use Average Precision (AP) (Everingham et al., 2015) to quantify the accuracy.

	Bicycle	Cat	Dog	Female Person	Lorry Truck	Male Person	Motor cycle	SUV	Van
Bicycle	99.92%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Cat	0.00%	99.86%	0.13%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Dog	0.00%	0.14%	99.87%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Female Person	0.00%	0.00%	0.00%	99.77%	0.00%	0.14%	0.00%	0.00%	0.00%
Lorry Truck	0.00%	0.00%	0.00%	0.00%	90.37%	0.00%	0.00%	0.00%	0.00%
Male Person	0.00%	0.00%	0.00%	0.23%	0.00%	99.86%	0.00%	0.00%	0.00%
Motor cycle	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	99.93%	0.00%	0.00%
SUV	0.00%	0.00%	0.00%	0.00%	0.14%	0.00%	0.00%	98.40%	0.94%
Van	0.08%	0.00%	0.00%	0.00%	9.49%	0.00%	0.07%	1.60%	99.06%

Figure 19: Confusion Matrix of Satyam Result on ImageNet-10

	Clap	Jump	Pick	Push	Run	Sit	Stand	Throw	Walk	Wave
Clap	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Jump	0.00%	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Pick	0.00%	0.00%	97.50%	0.00%	0.00%	0.00%	0.00%	2.13%	0.00%	0.00%
Push	0.00%	0.00%	0.00%	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
Run	0.00%	0.00%	0.00%	0.00%	97.56%	0.00%	0.00%	0.00%	0.00%	0.00%
Sit	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%	0.00%	0.00%	0.00%	0.00%
Stand	0.00%	0.00%	2.50%	0.00%	0.00%	0.00%	100.00%	0.00%	0.00%	0.00%
Throw	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	97.87%	0.00%	0.00%
Walk	0.00%	0.00%	0.00%	0.00%	2.44%	0.00%	0.00%	0.00%	100.00%	0.00%
Wave	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%

Figure 20: Confusion Matrix of Satyam Result on JHMDB

Figure 21: Example of counting error resulting from partially visible cars

Figure 22: Example of missing segmentation labels from PASCAL. From left to right: raw image, PASCAL label, Satyam label. Satyam segments a small truck on the top left corner which was not present in the ground truth.

We use a range of IoUs (0.5-0.95 with steps of 0.05) to compute the average to avoid a bias towards a specific value. Satyam achieves an AP of 90.03%. We also provide a match-precision of 94.77% and a match-recall of 94.56% using an IoU of 0.5. The dominant cause of false positives is missing annotations in the ground-truth. Figure 22 shows examples of such missing annotations from PASCAL that Satyam's users were able to produce. The primary cause of false negatives is that our experiments used a lower value of η_{cov} than appropriate for this task; we are rectifying this currently.

Object Tracking in Videos. A track is a sequence of bounding boxes across multiple frames. Consequently, we use the same match criterion for this template as detection across all the video frames. As seen from (Figure 17), Satyam has a match-precision and match-recall of around 95%. To understand why, we explored worker performance at different positions in the chunk: we found that, as workers get to the end of a chunk, they tend not to start tracking new objects. Decreasing the chunk size and increasing the overlap among consecutive chunks would increase accuracy, at higher cost.

6.3 Re-training Models using Satyam

A common use case for Satyam is fine-tuning ML models for improving their performance using data specific to a deployment (similar to example in Section 1). In this section, we validate this observation by showing that (Figure 23): a) retraining ML models using Satyam groundtruth outperforms off-the-shelf pre-trained models, and b) models retrained using Satyam groundtruth perform comparably with models retrained using benchmarks. When re-training and testing a model, either with Satyam or benchmark groundtruth, we use standard methodology to train on 80% of the data, and test on 20%. In all cases, we retrain the last layer using accepted methodology (Donahue et al., 2013; Retraining).

Image Classification. For this job category, we evaluate retraining a well-known state-of-the-art image classification neural network, Inception (V3) (Szegedy et al., 2016). The original model was pre-trained on ImageNet-1000 (Deng et al., 2009) for 1000 different classes of objects. Using this model as-is on ImageNet-10 yields a classification accuracy (F1-score (Goutte & Gaussier, 2005)) of about 60% (Figure 23). Retraining the models using the ImageNet-10 groundtruth increases their accuracy to 94.76%, while retraining on Satyam results in an accuracy of 95.46%.

Object Detection in Images. For this category, we evaluate YOLO (Redmon et al., 2015), pre-trained on the MS-COCO dataset. Our measure of accuracy is the mean average precision (Everingham et al., 2010), a standard metric for object detection and localization that combines precision and recall by averaging precision over all recall values. The pre-trained YOLO model has high (80%) mean average precision, but retraining it using KITTI-Object increases this to 90.1%. Retraining YOLO using Satyam groundtruth matches KITTI-Object's performance, with a mean average precision of 91.0% (Figure 23).

Tracking in Videos. As of this writing, the highest ranked open-source tracker on the KITTI Tracking Benchmark leaderboard is MDP (Xiang et al., 2015), so we evaluate this tracker (with YOLO as the underlying object detector) using the standard Multi-Object Tracking Accuracy (MOTA (Bernardin & Stiefelhagen, 2008)) metric, which also combines precision and recall. MDP using YOLO-CoCo's detections achieves a MOTA of 61.83% as depicted in Figure 23 but fine-tuning Yolo's last layer using the labels from KITTI and Satyam improve MOTA to 78% and 77.77% respectively. Further investigation reveals that the improvement in MoTA from fine-tuning was primarily due to improvement in recall – while precision was already high (98%) before fine-tuning, recall was only 63.54%. After fine-tuning recall improved to 81.70% for KITTI and 83.24% for Satyam.

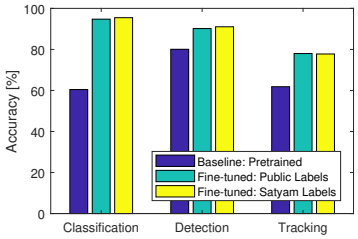

Figure 23: Training Performance of Satyam

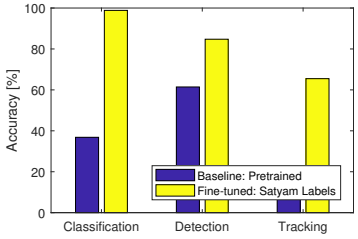

Figure 24: End to End Training using Satyam Labels

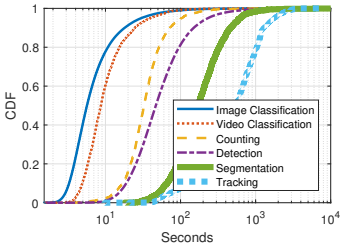

Figure 25: CDF of Time Spent Per Task of All Job Categories

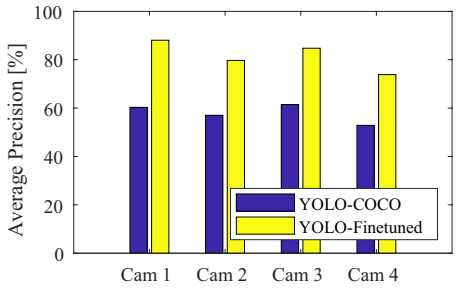

Figure 26: Improvement of performance with fine-tuned YOLO.

6.4 Satyam In Real-World Deployments

In order to evaluate the impact of using Satyam in the real world, we extracted images at 1 frame/minute from the video streams of 4 live HD quality traffic surveillance cameras (labeled SURV-1 to SURV-4), over one week (7 days) between 7:00 am and 7:00 pm each day. These cameras are located at major intersections in two U.S cities.

We now show that using Satyam groundtruth to fine-tune ML models can result in improved classification, detection, and tracking performance. For this, we use the SURV dataset, which has surveillance camera images from four intersections, to obtain ground-truth with Satyam, then re-trained YOLO-CoCo (Donahue et al., 2013; Retraining) with 80% of the ground-truth and tested on the remaining 20%.

Satyam re-training can improve YOLO-CoCo performance uniformly across the four surveillance cameras (Figure 26). The average precision improves from 52-61% for the pre-trained models to 73-88% for the fine-tuned models – an improvement of 20-28%. This validates our assertion that camera fine-tuning will be essential for practical deployments, motivating the need for a system like Satyam.

Figure 24 demonstrates that these benefits carry over to other job categories as well and shows fine-tuning Inception v3 for classification for one of the cameras, SURV-3. To compute this result, we used our groundtruth data from SURV-3 for the detection task, where workers also labeled objects, then trained Inception to focus on one object type, namely cars. While the pre-trained Inception model works poorly on SURV-3, fine-tuning the model results in an almost perfect classifier. Similarly, fine-tuning also results in an almost 40% improvement in the MOTA metric for the tracker.

6.5 Time-to-Completion and Cost

Figure 17 also shows the median time to complete an entire job, which ranges from 7 hours to 7 days. From this, we can derive the median latency per object, which ranges from 2.5 seconds/image for image classification to 125 seconds/object for tracking. That figure also shows the cost of annotating an object in *person-seconds/object*: the actual dollar figure paid is proportional to this (§6.6). By this metric (Figure 17), image and video classification, and counting cost few tens of person-seconds/object while detection, segmentation, and tracking require 160, 599, and 978 person-seconds respectively.

6.6 Price Adaptation

Figure 25 is a CDF of the times taken by workers for all the various job categories in our evaluation. It clearly shows that the time taken to complete a task can vary by 3 orders of magnitude across our job categories. Figure 27 depicts the pdf of the times taken for the same category – counting task – but for two different data sets *i.e.,* CARPK-1 and KITTI-Obj. KITTI-Obj has around 10 vehicles on average, and CARPK around 45 in each image, and the distribution of worker task completion times varies significantly across these datasets. (As an aside, both these figures have a long tail: we have seen several cases where workers start tasks but sometimes finish it hours later).

These differences motivate price adaptation. To demonstrate price adaptation in Satyam, we show the temporal evolution of price per HIT for CARPK-1 and KITTI-Obj in Figure 28. HIT price for KITTI-Obj converges within 200 results to the ideal target value (corresponding to median task completion time). CARPK-1 convergence is slightly slower due to its larger variability in task completion times (Figure 27).

6.7 Worker Filtering

To evaluate the efficacy of worker filtering ran Satyam with and without worker filtering turned on for each of the tem-

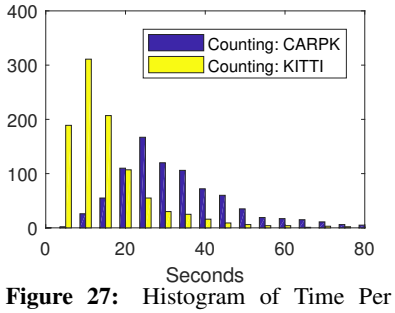

Figure 27: Histogram of Time Per Counting Task over Different Datasets

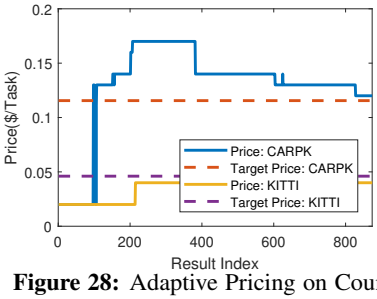

Figure 28: Adaptive Pricing on Counting Task

Web-UI Template	Original	After Filtering
Image Classification	89.0%	90.3%
Video Classification	88.7%	91.4%
Counting in Images	92.4%	94.0%
Detection in Images	75.0%	82.9%
Segmentation in Images	65.8%	81.4%
Tracking in Video	68.1%	86.6%

Figure 29: Approval Rates for various Satyam templates

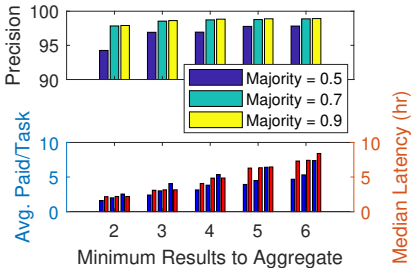

Figure 30: Accuracy, Latency and Cost: Image Classification

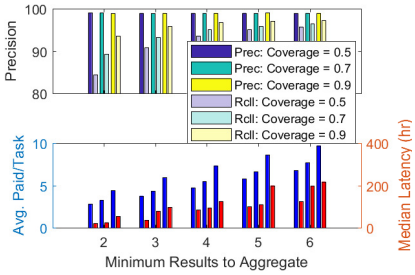

Figure 31: Accuracy, Latency and Cost: Detection

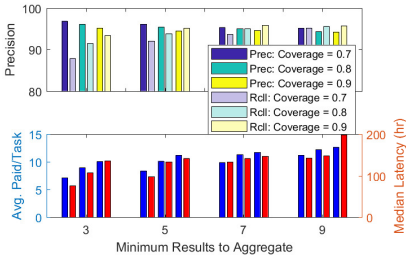

Figure 32: Accuracy, Latency and Cost: Tracking

plates. Figure 29 shows that for classification, counting and detection the approval rate is already quite high ranging close to 90% and thus worker filtering brings about a modest increase in approval rates. For more involved tasks such as tracking and segmentation, the approval rates show a dramatic increase from 60% to over 80%.

6.8 Parameter Sensitivity

Satyam's groundtruth fusion and result evaluation algorithms have several parameters and Figure 17 presents results for the best parameter choice. We have analyzed the entire space of parameters to determine the parameters that Satyam performance most crucially depends upon in terms of accuracy, latency and cost.

Image classification is sensitive only to two parameters: n_{min}, the minimum number of results before aggregation can commence, and β, the fraction determining the super-majority. The upper graph in Figure 30 shows how classification accuracy varies as a function of these two parameters. Because the cost and latency of groundtruth collection varies with parameters, the lower graph shows the cost (blue bars) and the latency (red bars) for each parameter choice. From this, we can see that when $n_{min} \geq 3$ and $\beta \geq 0.7$ the accuracy does not improve significantly, however, cost and latency increase. This indicates that $n_{min} = 3$ and $\beta = 0.7$ are good parameter choices, with high accuracy, while having moderate cost and latency.

We have conducted similar analyses for video classification, counting, object detection (Figure 31), segmentation, and tracking (Figure 32). Space constraints preclude a detailed discussion, but the key conclusions are: (a) All job categories are sensitive to n_{min}, the minimum number of results before Satyam attempts to aggregate results; (b) Each category is sensitive to one other parameter. For classification, this is the β parameter that determines the super-majority criterion. For counting, it is the error tolerance ϵ. For detection and tracking, it is η_{cov}, the fraction of corroborated groundtruth elements; and (c) In each case, there exists a parameter settings at which provides good groundtruth performance at moderate cost and latency.

7 Related Work

Image recognition using crowdtasking. ImageNet training data for classification was generated using AMT, and uses majority voting for consensus (Deng et al., 2009). Prior work (Su et al., 2012) has also shown crowdtasking to be successful for detection: unlike Satyam, in this work, quality control is achieved by using workers to rate other workers, and majority voting picks the best bounding box. These use one-off systems to automate HIT management and consensus, but do not consider payment management. Satyam achieves comparable performance to these systems but supports more vision tasks. Third party commercial crowdtasking systems exist to collect groundtruth for machine vision (Figure Eight; Spare5). Other approaches have developed one-off systems built on top of AMT for more complex vision tasks, including feature generation for

sub-class labeling (Deng et al., 2013), and sentence-level textual descriptions (Krishna et al., 2017; Parameswaran et al., 2014): more generally, future machine vision systems will need annotated groundtruth for other complex annotations including scene characterization, activity recognition, visual story-telling (Kovashka et al., 2016) and we have left it to future work to extend Satyam to support these.

Crowdtasking cost, quality, and latency. Prior work has extensively used multiple worker annotations and majority voting to improve quality (Deng et al., 2009; Su et al., 2012). For binary classification tasks in a one-shot setting, lower cost solutions exist to achieve high quality (Karger et al., 2011) or low latency (Krishna et al., 2016). For top-k classification (e.g., finding the k least blurred images in a set) several algorithms can be used for improving crowdtasking consensus (Zhang et al., 2016). Other work has explored this cost-quality tradeoff (Garcia-Molina et al., 2016) in different crowd-tasking settings: de-aliasing entity descriptions (Verroios et al., 2017; Khan & Garcia-Molina, 2016), or determining answers to a set of questions (Khan & Garcia-Molina, 2017). Satyam devises novel automated consensus algorithms for image recognition tasks based on the degree of pixel overlap between answers.

Crowdtasking platforms. Many marketplaces put workers in touch with requesters for freelance work (Guru; Freelancer; 99designs), for coders (TopCoder), for software testing (Mob4Hire; uTest), or for generic problem solving (Innocentive). Satyam adds automation on top of an existing generic marketplace, AMT. Other systems add similar kinds of automation, but for different purposes. Turkit (Little et al., 2009) and Medusa (Ra et al., 2012) provide an imperative high-level programming language for human-in-the-loop computations and sensing respectively. Collaborative crowdsourcing (Ikeda et al., 2016) automates the decomposition of more complex tasks into simpler ones, and manages their execution.

8 Conclusions

In this paper, we have presented Satyam, a cloud-based platform for automating large-scale groundtruth collection for machine vision applications. Satyam's groundtruth matches that of existing ML benchmarks datasets, ML models retrained with Satyam are as good as those re-trained with benchmark datasets, and ML models fine-tuned with Satyam's groundtruth improve detection accuracy by up to 28% in real deployments over pre-trained models.

References

Amazon Mechanical Turk. https://www.mturk.com/, 2018.

99designs. 99designs. https://99designs.com/.

Bernardin, K. and Stiefelhagen, R. Evaluating multiple object tracking performance: The clear mot metrics. J. Image Video Process., 2008:1:1–1:10, January 2008. ISSN 1687-5176. doi: 10.1155/2008/246309. URL http://dx.doi.org/10.1155/2008/246309.

Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255, June 2009. doi: 10.1109/CVPR.2009.5206848.

Deng, J., Krause, J., and Fei-Fei, L. Fine-grained crowdsourcing for fine-grained recognition. In Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition, CVPR '13, pp. 580–587, Washington, DC, USA, 2013. IEEE Computer Society. ISBN 978-0-7695-4989-7. doi: 10.1109/CVPR.2013.81. URL https://doi.org/10.1109/CVPR.2013.81.

Detection. Samples of Object Detection Groundtruth Using Satyam. https://www.dropbox.com/sh/qs8peao1k88a9ya/AADMFJ6EL7yE7WKZEd5bskmza?dl=0, 2018.

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. Decaf: A deep convolutional activation feature for generic visual recognition. CoRR, abs/1310.1531, 2013. URL http://arxiv.org/abs/1310.1531.

Everingham, M., Gool, L., Williams, C. K., Winn, J., and Zisserman, A. The pascal visual object classes (voc) challenge. Int. J. Comput. Vision, 88(2):303–338, June 2010. ISSN 0920-5691. doi: 10.1007/s11263-009-0275-4. URL http://dx.doi.org/10.1007/s11263-009-0275-4.

Everingham, M., Eslami, S. M. A., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. The pascal visual object classes challenge: A retrospective. International Journal of Computer Vision, 111(1):98–136, January 2015.

Figure Eight. Figure Eight. https://www.figure-eight.com/.

Freelancer. Freelancer. https://www.freelancer.com/info/how-it-works.

Garcia-Molina, H., Joglekar, M., Marcus, A., Parameswaran, A., and Verroios, V. Challenges in data crowdsourcing. IEEE Transactions on Knowledge & Data Engineering, 28(4):901–911, April 2016. ISSN 1041-4347. doi: 10.1109/TKDE.2016.2518669. URL doi.ieeecomputersociety.org/10.1109/TKDE.2016.2518669.

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

Goutte, C. and Gaussier, E. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In Proceedings of the 27th European Conference on Advances in Information Retrieval Research, ECIR'05, pp. 345–359, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3-540-25295-9, 978-3-540-25295-5. doi: 10.1007/978-3-540-31865-1_25. URL http://dx.doi.org/10.1007/978-3-540-31865-1_25.

Guru. Guru. https://www.guru.com/.

Hsieh, M.-R., Lin, Y.-L., and Hsu, W. H. Drone-based object counting by spatially regularized regional proposal networks. In The IEEE International Conference on Computer Vision (ICCV). IEEE, 2017.

Ikeda, K., Morishima, A., Rahman, H., Roy, S. B., Thirumuruganathan, S., Amer-Yahia, S., and Das, G. Collaborative crowdsourcing with crowd4u. Proc. VLDB Endow., 9(13):1497–1500, September 2016. ISSN 2150-8097. doi: 10.14778/3007263.3007293. URL https://doi.org/10.14778/3007263.3007293.

Innocentive. Innocentive. https://www.innocentive.com/.

Jhuang, H., Gall, J., Zuffi, S., Schmid, C., and Black, M. J. Towards understanding action recognition. In International Conf. on Computer Vision (ICCV), pp. 3192–3199, December 2013.

Karger, D. R., Oh, S., and Shah, D. Iterative learning for reliable crowdsourcing systems. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F., and Weinberger, K. Q. (eds.), Advances in Neural Information Processing Systems 24, pp. 1953–1961. Curran Associates, Inc., 2011. URL http://papers.nips.cc/paper/4396-iterative-learning-for-reliable-crowdsourcing-systems.pdf.

Khan, A. R. and Garcia-Molina, H. Attribute-based crowd entity resolution. In Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, CIKM '16, pp. 549–558, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4073-1. doi: 10.1145/2983323.2983831. URL http://doi.acm.org/10.1145/2983323.2983831.

Khan, A. R. and Garcia-Molina, H. Crowddqs: Dynamic question selection in crowdsourcing systems. In Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD '17, pp. 1447–1462, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4197-4. doi: 10.1145/3035918.3064055. URL http://doi.acm.org/10.1145/3035918.3064055.

Kovashka, A., Russakovsky, O., Fei-Fei, L., and Grauman, K. Crowdsourcing in computer vision. CoRR, abs/1611.02145, 2016. URL http://arxiv.org/abs/1611.02145.

Krishna, R., Hata, K., Chen, S., Kravitz, J., Shamma, D. A., Li, F., and Bernstein, M. S. Embracing error to enable rapid crowdsourcing. CoRR, abs/1602.04506, 2016. URL http://arxiv.org/abs/1602.04506.

Krishna, R., Zhu, Y., Groth, O., Johnson, J., Hata, K., Kravitz, J., Chen, S., Kalantidis, Y., Li, L.-J., Shamma, D. A., Bernstein, M. S., and Fei-Fei, L. Visual genome: Connecting language and vision using crowdsourced dense image annotations. International Journal of Computer Vision, 123(1):32–73, May 2017.

Little, G., Chilton, L. B., Goldman, M., and Miller, R. C. Turkit: Tools for iterative tasks on mechanical turk. In Proceedings of the ACM SIGKDD Workshop on Human Computation, HCOMP '09, pp. 29–30, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-672-4. doi: 10.1145/1600150.1600159. URL http://doi.acm.org/10.1145/1600150.1600159.

Mob4Hire. Mob4Hire. https://www.mob4hire.com/.

MTurk SDK. AWS MTurk SDK. https://www.nuget.org/packages/AWSSDK.MTurk/, 2018.

mturk-spam. Mechanical Turk: Now with 40.92% Spam. http://www.behind-the-enemy-lines.com/2010/12/mechanical-turk-now-with-4092-spam.html.

Parameswaran, A., Teh, M. H., Garcia-Molina, H., and Widom, J. Datasift: A crowd-powered search toolkit. In Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, SIGMOD '14, pp. 885–888, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2376-5. doi: 10.1145/2588555.2594510. URL http://doi.acm.org/10.1145/2588555.2594510.

Ra, M.-R., Liu, B., La Porta, T. F., and Govindan, R. Medusa: A programming framework for crowd-sensing applications. In Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, MobiSys '12, pp. 337–350, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1301-8. doi: 10.1145/2307636.2307668. URL http://doi.acm.org/10.1145/2307636.2307668.

Redmon, J., Divvala, S. K., Girshick, R. B., and Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. CoRR, abs/1506.02640, 2015. URL http://arxiv.org/abs/1506.02640.

Retraining. How to Retrain an Image Classifier for New Categories. https://www.tensorflow.org/tutorials/image_retraining.

Satyam Portal. Satyam Portal. https://satyamresearchportal.azurewebsites.net, 2018.

Spare5. Spare5 Website. https://app.spare5.com/fives, 2018.

Su, H., Deng, J., and Fei-Fei, L. Crowdsourcing annotations for visual object detection. In The 4th Human Computation Workshop, HCOMP@AAAI 2012, Toronto, Ontario, Canada, July 23, 2012., 2012.

Surowiecki, J. The Wisdom of Crowds. Anchor, 2005. ISBN 0385721706.

Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. A. Inception-v4, inception-resnet and the impact of residual connections on learning. In ICLR 2016 Workshop, 2016. URL https://arxiv.org/abs/1602.07261.

TopCoder. TopCoder. https://www.topcoder.com/.

Tracking. Samples of Tracking Groundtruth Using Satyam. https://www.dropbox.com/sh/mcdsadqmk91hbgc/AAAdMTL45YuuOJYM7Pek13Rea?dl=0, 2018.

uTest. uTest. https://www.utest.com/.

Vatic. Vatic: Video annotation tool from irvine, ca. https://github.com/cvondrick/vatic.

Verroios, V., Garcia-Molina, H., and Papakonstantinou, Y. Waldo: An adaptive human interface for crowd entity resolution. In Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD '17, pp. 1133–1148, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4197-4. doi: 10.1145/3035918.3035931. URL http://doi.acm.org/10.1145/3035918.3035931.

Xiang, Y., Alahi, A., and Savarese, S. Learning to Track: Online Multi-Object Tracking by Decision Making. In Proceedings of the IEEE International Conference on Computer Vision, pp. 4705–4713, 2015.

YACVID. Yet Another Computer Vision Index To Datasets (YACVID). http://riemenschneider.hayko.at/vision/dataset/.

Zhang, X., Li, G., and Feng, J. Crowdsourced top-k algorithms: An experimental evaluation. Proc. VLDB Endow., 9(8):612–623, April 2016. ISSN 2150-8097. doi: 10.14778/2921558.2921559. URL http://dx.doi.org/10.14778/2921558.2921559.