Surpassing Humans and Computers with JELLYBEAN: Crowd-Vision-Hybrid Counting Algorithms

Akash Das Sarma
Stanford University
akashds@stanford.edu

Ayush Jain
University of Illinois
ajain42@illinois.edu

Arnab Nandi
The Ohio State University
arnab@cse.osu.edu

Aditya Parameswaran
University of Illinois
adityagp@illinois.edu

Jennifer Widom
Stanford University
widom@cs.stanford.edu

Abstract

Counting objects is a fundamental image processisng primitive, and has many scientific, health, surveillance, security, and military applications. Existing supervised computer vision techniques typically require large quantities of labeled training data, and even with that, fail to return accurate results in all but the most stylized settings. Using vanilla crowdsourcing, on the other hand, can lead to significant errors, especially on images with many objects. In this paper, we present our JellyBean suite of algorithms, that combines the best of crowds and computer vision to count objects in images, and uses judicious decomposition of images to greatly improve accuracy at low cost. Our algorithms have several desirable properties: *(i)* they are *theoretically optimal or near-optimal*, in that they ask as few questions as possible to humans (under certain intuitively reasonable assumptions that we justify in our paper experimentally); *(ii)* they operate under *stand-alone* or *hybrid* modes, in that they can either work independent of computer vision algorithms, or work in concert with them, depending on whether the computer vision techniques are available or useful for the given setting; *(iii)* they *perform very well* in practice, returning accurate counts on images that no individual worker or computer vision algorithm can count correctly, while not incurring a high cost.

1 Introduction

The field of computer vision (Forsyth and Ponce 2003; Szeliski 2010) concerns itself with the understanding and interpretation of the contents of images or videos. Many of the fundamental problems in this field are far from solved, with even the state-of-the-art techniques achieving poor results on benchmark datasets. For example, the recent techniques for image categorization achieve average precision ranging from 19.5% (for the `chair` class) to 65% (for the `airplane` class) on a canonical benchmark (Everingham et al. 2014).

Counting is one such fundamental image understanding problem, and refers to the task of counting the number of items of a particular type within an image or video.

Counting is important. Counting objects in images or videos is a ubiquitous problem with many applications. For instance, biologists are often interested in counting the number of cell colonies in periodically captured photographs of petri dishes; counting the number of individuals at concerts or demonstrations is often essential for surveillance and security (Liu et al. 2005); counting nerve cells or tumors is standard practice in medical applications (Loukas et al. 2003); and counting the number of animals in photographs of ponds or wildlife sanctuaries is often essential for animal conservation (Russell et al. 1996). In many of these scenarios, making errors in counting can have unfavorable consequences. Furthermore, counting is a prerequisite to other, more complex computer vision problems requiring a deeper, more complete understanding of images.

Counting is hard for computers. Unfortunately, current supervised computer vision techniques are typically very poor at counting for all but the most stylized settings, and cannot be relied upon for making strategic decisions. The computer vision techniques primarily have problems with *occlusion*, i.e., identifying objects that are partially hidden behind other objects. As an example, consider Figure 1, depicting the performance of a recent pre-trained face detection algorithm (Zhu and Ramanan 2012). The algorithm performs poorly for occluded faces, detecting only 35 out of 59 (59.3%) faces. The average precision for the state-of-the-art person detector is only 46% (Everingham et al. 2014). Furthermore, these techniques are not generalizable; separate models are needed for each new application. For instance, if instead of wanting to count the number of faces in a photograph, we needed to count the number of women, we would need to start afresh by training an entirely new model.

Even humans have trouble counting. While humans are much better at counting than automated techniques, and are good at detecting occluded (hidden) objects, as the number of objects in the image increases, they start making mistakes. To observe this behavior experimentally, we had workers count the number of cell colonies in simulated fluorescence microscope images with a wide range of counts. We plot the results in Figure 2, displaying the average error in count (on

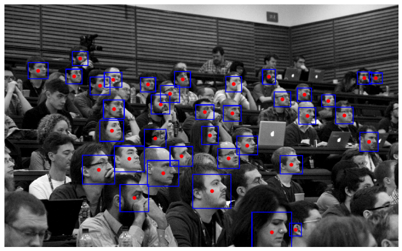


Figure 1: Challenging image for Machine Learning

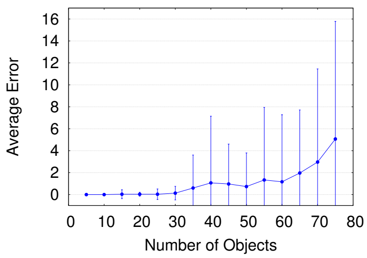


Figure 2: Worker Error

the y-axis) versus the actual count (on the x-axis). As can be seen in the figure, crowd workers make few mistakes until the number of cells hit 20 or 25, after which the average error increases. In fact, when the number of cells reaches 75, the average error in count is as much as 5. (There are in fact many images with even higher errors.) Therefore, simply showing each image to one or more workers and using those counts is not useful if accurate counts are desired.

The need for a hybrid approach. Thus, since both humans and computers have trouble with counting, there is a need for an approach that best *combines human and computer capabilities to count accurately while minimizing cost.* These techniques would certainly help with the counting problem instance at hand—the alternative of having a biology, security, medical, or wildlife expert count objects in each image can be error-prone and costly. At the same time, these techniques would also *enable the collection of training data at scale*, and thereby spur the generation of even more capable computer vision algorithms. To the best of our knowledge, we are the first to articulate and make concrete steps towards solving this important, fundamental vision problem.

Key idea: judicious decomposition. Our approach, inspired by Figure 2, is to judiciously decompose an image into smaller ones, focusing worker attention on the areas that require more careful counting. Since workers have been observed to be more accurate on images with fewer objects, the key idea is to obtain reliable counts on smaller, targeted sub-images, and use them to infer counts for the original image. However, it is not clear when or how we should divide an image, or where to focus our attention by assigning more workers. For example, we cannot tell a-priori if all the cell colonies are concentrated in the upper left corner of the image. Another challenge is to divide an image while being cognizant of the fact that you may cut across objects during the division. This could result in double-counting some objects across different sub-images.

Adaptivity to two modes. In the spirit of combining the best of human worker and computer expertise, when available, we develop algorithms that are near-optimal for two separate regimes or modes:

- First, assuming we have no computer vision assistance (i.e., no prior computer vision algorithm that could guide us to where the objects are in the image), we design an algorithm that will allow us to *narrow our focus* to the right portions of the image requiring special attention. The algorithm, while intuitively simple to describe, is *theoretically optimal in that it achieves the best possible competitive ratio*, under certain assumptions. At the same time, in practice, on a real crowd-counting dataset, the cost of our algorithm is within $2.75\times$ *of the optimal "oracle" algorithm that has perfect information*, while still maintaining very high accuracy.
- Second, if we have primitive or preliminary computer vision algorithms that provide segmentation and prior count information, we design algorithms that can use this knowledge to once again identify the regions of the image to focus our resources on, by "fast-forwarding" to the right areas. We formulate the problem as a *graph binning problem, known to be* NP-COMPLETE *and provide an efficient* articulation-point based heuristic for this problem. We show that in practice, our algorithm has a very high accuracy, and only incurs $1.3\times$ *the cost of the optimal, perfect information "oracle" algorithm.*

We dub our algorithms for these two regimes as the *Jelly-Bean* algorithm suite, as a homage to one of the early applications of crowd counting[1].

Here is the outline for the rest of the paper as well as our contributions (We describe related work in Section 6.)

- We model images as trees with nodes representing image segments and edges representing image-division. Given this model, we present a novel formulation of the counting problem as a search problem over the nodes of the tree (Section 2).
- We present a crowdsourced solution to the problem of counting objects over a given image-tree. We show that under reasonable assumptions, our solution is provably optimal (Section 3).
- We extend the above solution to a *hybrid* scheme that can work in conjunction with computer vision algorithms, leveraging prior information to reduce the cost of the crowdsourcing component of our algorithm, while significantly improving our count estimates (Section 4).
- We validate the performance of our algorithms against credible baselines using experiments on real data from two different representative applications (Section 5).

For readers interested in finer details and detailed evaluations, we also provide an extended technical report (Sarma et al. 2015).

[1]Counting or estimating the number of jellybeans in a jar has been a popular activity in fairs since 1900s, while also serving as unfortunate vehicle for disenfranchisement (NBC News 2005).

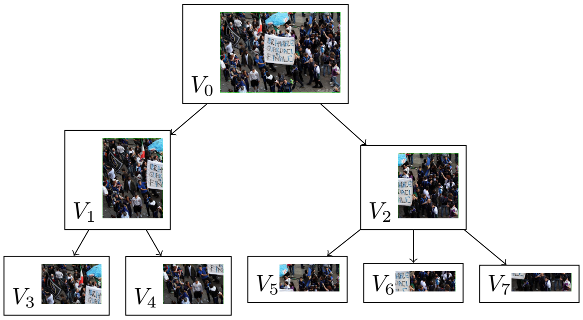


Figure 3: Segmentation Tree

2 Preliminaries

In this section, we describe our data model for the input images and our interaction model for worker responses.

2.1 Data Model

Given an image with a large number of (possibly heterogenous) objects, our goal is to estimate, with high accuracy, the number of objects present. As noted above in Figure 2, humans can accurately count up to a small number of objects, but make significant errors on images with larger numbers of objects. To reduce human error, we split the image into smaller portions, or *segments*, and ask workers to estimate the number of objects in each segment. Naturally, there are many ways we may split an image. We discuss our precise algorithms for splitting an image into segments subsequently. For now, we assume that the segmentation is fixed.

We represent a given image and all its segments in the form of a directed tree $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, called a *segmentation tree*. The original image is the root node, V_0, of the tree. Each node $V_i \in \mathbf{V}, i \in \{0, 1, 2, \dots\}$ corresponds to a sub-image, denoted by $\texttt{Image}(V_i)$. We call a node V_j a *segment* of V_i if $\texttt{Image}(V_j)$ is contained in $\texttt{Image}(V_i)$. A directed edge exists between nodes V_i and $V_j : (V_i, V_j) \in \mathbf{E}$, if and only if V_i is the lowest node in the tree, i.e. smallest image, such that V_j is a segment of V_i. For brevity, we refer to the set of children of node V_i (denoted as $\mathbf{C}_i$) as the split of V_i. If $\mathbf{C}_i = \{V_1, \cdots, V_s\}$, we have $\texttt{Image}(V_i) = \bigcup_{j \in \{1, \cdots, s\}} \texttt{Image}(V_j)$.

For example, consider the segmentation tree in Figure 3. The original image, V_0, can be split into the two segments $\{V_1, V_2\}$. V_1, in turn, can be split into segments $\{V_3, V_4\}$. Intuitively, the root image can be thought of as physically segmented into the five leaf nodes $\{V_3, V_4, V_5, V_6, V_7\}$.

We assume that all segments of a node are non-overlapping. That is, given any node V_i and its immediate set of children $\mathbf{C}_i$, we have (1) $\texttt{Image}(V_i) = \bigcup_{V_j \in \mathbf{C}_i} \texttt{Image}(V_j)$ and (2) $\texttt{Image}(V_j) \cap \texttt{Image}(V_k) = \phi \, \forall \, V_j, V_k \in \mathbf{C}_i$ We denote the actual number of objects in a segment, $\texttt{Image}(V_i)$, by $\texttt{TrueCount}(V_i)$. Our assumption of non-overlapping splits ensures that $\texttt{TrueCount}(V_i) = \sum_{V_j \in \mathbf{C}_i} \texttt{TrueCount}(V_j)$.

One of the major challenges of the counting problem is to estimate these $\texttt{TrueCount}$ values with high accuracy, by using elicited worker responses. Given the segmentation tree $\mathbf{G}$ for image V_0, we can ask workers to count, possibly multiple times, the number of objects in any of the segments. For example, in Figure 3, we can ask workers to count the number of objects in the segments (V_3), (V_4), (V_5), (V_6), (V_7), (V_1), (V_2),(V_0). While we can obtain counts for different nodes in the segmentation tree, we need to consolidate these counts to a final estimate for V_0. To help with this, we introduce the idea of a *frontier*, which is central to all our algorithms. Intuitively, a frontier F is a set of nodes whose corresponding segments do not overlap, and cover the entire original image, $\texttt{Image}(V_0)$ on merging. We formally define this notion below.

Definition 2.1 (Frontier). *Let $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ be a segmentation tree with root node V_0 . A set of k nodes given by $F = \{V_1, V_2, \dots, V_k\}$, where $V_i \in \mathbf{V} \, \forall \, i \in \{1, \dots, k\}$ is a frontier of size k if $\texttt{Image}(V_0) = \bigcup_{V_i \in F} \texttt{Image}(V_i)$, and $\texttt{Image}(V_i) \cap \texttt{Image}(V_j) = \phi \, \forall \, V_i, V_j \in F$*

A frontier F is now a set of nodes in the segmentation tree such that taking the sum of $\texttt{TrueCount}(\cdot)$ over these nodes returns the desired count estimate $\texttt{TrueCount}(V_0)$. Continuing with our example in Figure 3, we have the following five possible frontiers: $\{V_0\}$, $\{V_1, V_2\}$, $\{V_1, V_5, V_6, V_7\}$, $\{V_2, V_3, V_4\}$, and $\{V_3, V_4, V_5, V_6, V_7\}$.

2.2 Worker Behavior Model

Intuitively, workers estimate the number of objects in an image correctly if the image has a small number of objects. As the number of objects increases, it becomes difficult for humans to keep track of which objects have been counted. Based on the experimental evidence in Figure 2, we hypothesize that there is a threshold number of objects, above which workers start to make errors, and below which their count estimates are accurate. Let this threshold be d^*. So, in our interface, we ask the workers to count the number of objects in the query image. If their estimate, d, is less than d^*, they provide that estimate. If not, they simply inform us that the number of objects is greater than d^*. This allows us to cap the amount of work done by the workers– the workers can count as far as they are willing to correctly, and if the number of objects is, say, in the thousands, they may just inform us that this is greater than d^* without expending too much effort. We denote the worker's estimate of $\texttt{TrueCount}(V)$ by $\texttt{WorkerCount}(V)$.

$$\texttt{WorkerCount}(V) = \begin{cases} \texttt{TrueCount}(V) & : \texttt{TrueCount}(V) \leq d^* \\ > d^* & : \texttt{TrueCount}(V) > d^* \end{cases}$$

Based on Figure 2, the threshold d^* is 20. We provide further experimental verification for this error model in (Sarma et al. 2015). While we could choose to use more complex error models, we find that the above model is easy to analyze and experimentally valid, and therefore suffices for our purposes.

3 Crowdsourcing-Only Solution

In this section, we consider the case when we do not have a computer vision algorithm at our disposal. Thus, we must use only crowdsourcing to estimate image counts. Since it is often hard to train computer vision algorithms for every new type of object, this is a scenario that often occurs in practice.

As hinted at in Section 2, the idea behind our algorithms is simple: we ask workers to estimate the count at nodes of the segmentation tree in a top-down expansion, until we reach a frontier such that we have a high confidence in the worker estimates for all nodes in that frontier.

3.1 Problem Setup

We are given a fixed b-ary segmentation tree i.e. a tree with each non-leaf node having exactly b children. We also assume that each object is present in exactly one segment across siblings of a node, and that workers follow the behavior model from Section 2.2. Some of these assumptions may not always hold in practice, and we discuss their relaxations later in Section 3.4.

For brevity, we will refer to displaying an image segment (node in the segmentation tree) and asking a worker to estimate the number of objects, as *querying the node*. Our problem can be therefore restated as that of *finding the exact number of objects in an image by querying as few nodes of the segmentation tree as possible*. Next, we describe our algorithm on this setting in Section 3.2, and give complexity and optimality guarantees in Section 3.3.

3.2 The FrontierSeeking Algorithm

Our algorithm is based on the simple idea that to estimate the number of objects in the root node, we need to find a frontier with all nodes having fewer than d^* objects. This is because the elicited `WorkerCounts` are trustworthy only at the nodes that meet this criteria. We call such a frontier a *terminating frontier*. If we query all nodes in such a terminating frontier, then the sum of the worker estimates on those nodes is in fact the correct number of objects in the root node, given our model of worker behavior.

If a node V has greater than d^* objects, then we cannot estimate the number of objects in its parent node, and consequently the root node, without querying V's children. Our algorithm, `FrontierSeeking`$(\mathbf{G})$, depends on this observation for finding a terminating frontier efficiently, and correspondingly obtaining a count for the root node, V_0. The algorithm simply queries nodes in a top-down expansion of the segmentation tree, for example, with a breadth-first or depth-first search. For each node, we query its children if and only if workers report its count as being higher than the threshold d^*. We continue querying nodes in the tree, only stopping our expansion at nodes whose counts are reported as smaller than d^*, until we have queried all nodes in a terminating frontier. We return the sum of the reported counts of nodes in this terminating frontier as our final estimate.

3.3 Guarantees

We now discuss the guarantees that our `FrontierSeeking` algorithm provides under our proposed model. Given an image and its segmentation tree, let F^* be a terminating frontier of the smallest size, having k nodes. Our goal is to find a terminating frontier with as few queried nodes as possible.

First, we note that any algorithm needs to query at least k nodes to get a true count of the number of objects in the given image. This follows trivially from the observation that we need to query at least one complete terminating frontier to obtain a count for the root node of the tree. To quantify the performance of our algorithm, we use a *competitive ratio* analysis (Borodin 1998). Intuitively, the competitive ratio of an algorithm is a measure of its worst case performance against the optimal oracle algorithm. Let $A(\mathbf{G})$ denote the sequence of questions asked, or nodes queried, by our online deterministic `FrontierSeeking` algorithm on segmentation graph $\mathbf{G}$. Let $|A(\mathbf{G})|$ be the corresponding number of questions asked. For the optimal oracle algorithm OPT, $|OPT(\mathbf{G})| = k$ where k is the size of the minimal terminating frontier of $\mathbf{G}$.

We now state the competitive ratio CR of our `FrontierSeeking` algorithm A_{FS} in the following theorem. The proofs can be found in (Sarma et al. 2015).

Theorem 3.1. *Let $\mathbb{G}$ be the set of all segmentation trees with fanout b. We have,* $\mathtt{CR}(A_{FS}) = \max_{\mathbf{G}\in\mathbb{G}} \frac{|A_{FS}(\mathbf{G})|}{|OPT(\mathbf{G})|} = \frac{b}{b-1}$.

The following theorem (combined with the previous one) states that our algorithm achieves the best possible competitive ratio across all online deterministic algorithms.

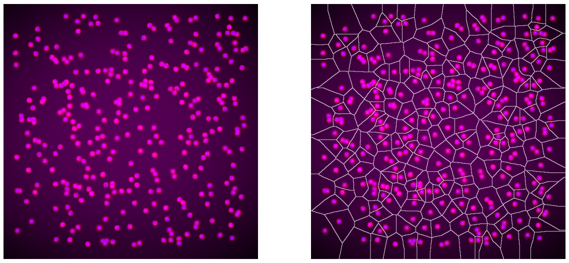
Theorem 3.2. *Let A be any online deterministic algorithm that computes the correct count for every given input segmentation tree G with fanout b. Then,* $\mathtt{CR}(A) \geq (\frac{b}{b-1})$.

3.4 Practical Setup

In this section we discuss some of the practical design challenges faced by our algorithm and give a brief overview of our current mechanisms for addressing these challenges.

Worker error. So far, we have assumed that human worker counts are accurate for nodes with fewer than d^* objects permitting us to query each node just a single time. However, this is not always the case in practice. In reality, workers may make mistakes while counting images with any number of objects (and we see this manifest in our experiments as well). So, in our algorithms, we show each image or node to multiple (five in our experiments) workers and aggregate their answers via `median` to obtain a count estimate for that node. We observe that although individual workers can make mistakes, our aggregated answers satisfy our assumptions in general (e.g., that the aggregate is always accurate when the count is less than d^*). While we use a primitive aggregation scheme in this work, it remains to be seen if more advanced aggregation schemes, such as those in (Karger, Oh, and Shah 2011; Parameswaran et al. 2012; Sheng, Provost, and Ipeirotis 2008) would lead to better results; we plan to explore these schemes in future work.

Segmentation tree. So far, we have also assumed that a segmentation tree with fanout b is already given to us. In practice, we are often only given the whole image, and have to create the segmentation tree ourselves. In our setup, we create a binary segmentation tree ($b = 2$) where the children of any node are created by splitting the parent into two halves along its longer dimension. As we will see later on, this choice leads to accurate results. While our algorithms also apply to segmentation trees of any fanout; further investigation is needed to study the effect of b on the cost and accuracy of the results.



(a) (b)

Figure 4: Biological image (a) before and (b) after partitioning

Segment boundaries. We have assumed that objects do not cross segmentation boundaries, i.e., each object is present in exactly one leaf node, and cannot be partially present in multiple siblings. Our segmentation does not always guarantee this. To handle this corner case, in our experiments we specify the notion of a "majority" object to workers with the help of an example image, and ask them to only count an object for an image segment *if the majority of it is present in that segment*. Once again, we find that this leads to accurate results in our present experiments. That said, we plan to explore more principled methods for counting partial objects in future work. For instance, one method could be to have workers separately count objects that are completely contained in a displayed image, and objects that cross a given number of segment boundaries.

We revisit these design decisions in Section 5.

4 Incorporating Computer Vision

Unlike the previous section, where we assumed a fixed segmentation tree, here, we use computer vision techniques (when easily available) to help build the segmentation tree, and use crowds to subsequently count segments in this tree. For certain types of images, existing machine learning techniques give two things: (1) a partitioning of the given image such that no object is present in multiple partitions, and (2) a prior count estimate of the number of objects in each partition. While these prior counts are not always accurate and still need to be verified by human workers, they allow us to skip some nodes in the implicit segmentation tree and "fast-forward" to querying lower nodes, thereby requiring fewer human tasks.

4.1 Partitioning

As a running example, we consider the application of counting cells in biological images. Figure 4a shows one such image, generated using SIMCEP, a tool for simulating fluorescence microscopy images of cell populations (Lehmussola et al. 2007). SIMCEP is the gold standard for testing algorithms in medical imaging, providing many tunable parameters that can simulate realworld conditions. We implement one simple partitioning scheme that splits any given such cell population image into many small, disjoint partitions. Applying this partitioning scheme to the image in Figure 4a yields Figure 4b. Combined with the partitioning scheme above, we can leverage existing machine learning (ML) techniques to estimate the number of objects in each of the partitions. We denote these ML-estimated counts on each partition, u, as *prior counts* or simply priors, d^u. Note that these priors are only approximate estimates, and still need to be verified by workers. We discuss details of our partitioning algorithm, prior count estimation, and other implementation details later in Section 4.3.

We use these generated partitions and prior counts to define a *partition graph* as follows:

Definition 4.1 (Partition Graph). *Given an image split into the set of partitions, V_P, we define its* partition graph*, $G_P = (V_P, E_P)$, as follows. Each partition, $u \in V_P$, is a node in the graph and has a weight associated with it equal to the prior, $w(u) = d^u$. Furthermore, an undirected edge exists between two nodes, $(u, v) \in E_P$, in the graph if and only if the corresponding partitions, u, v, are adjacent in the original image.*

Notice that while we have used one partitioning scheme and one prior count estimation technique for our example here, other machine learning or vision algorithms for this, as well as other settings provide similar information that will allow us to generate similar partition graphs. Thus, the setting where we begin with a partition graph is *general*, and applies to other scenarios.

Now, given a partition graph, one approach to counting the number of objects in the image could be to have workers count each partition individually. The number of partitions in a partition graph is, however, typically very large, making this approach impractical. For instance, most of the 5–6 partitions close to the lower right hand corner of the image above have precisely one cell, and it would be wasteful and expensive to ask a human to count each one individually. Next, we discuss an algorithm to merge these partitions into a smaller number, to minimize the number of human tasks.

4.2 Merging Partitions

Given a partition graph corresponding to an image, we leverage the prior counts on partitions to avoid the top-down expansion of segmentation trees described in Section 3. Instead, we infer the count of the image by merging its partitions together into a small number of *bins*, each of which can be reasonably counted by workers, and aggregating the counts across bins.

Merging problem. Intuitively, the problem of merging partitions is equivalent to identifying connected components (or bins) of the partition graph, with total weight (or count) at most d^*. Since workers are accurate on images with size up to d^*, we can then elicit worker counts for our merged components and aggregate them to find the count of the whole image. Overall, we have the following problem:

Problem 4.1 (Merging Partitions). *Given a partition graph $G_P = (V_P, E_P)$ of an image, partition the graph into k* disjoint *connected components in G_P, such that the sum of node weights in each component is less than or equal to d^*, and k is as small as possible.*

Enforcing disjoint components ensures that no components overlap over a common object, thereby avoiding double-counting. Furthermore, restricting our search to connected components ensures that our displayed images are contiguous — this is a desirable property for images displayed to workers over most applications, because it provides useful, necessary context to understand the image.

Hardness and reformulation. The solution to the above problem would give us the required merging. However, the problem described above can be shown to be NP-Complete, using a reduction from the NP-Complete problem of partitioning planar bipartite graphs (Dyer and Frieze 1985); our setting uses arbitrary planar graphs, and so our problem is more general. Thus, we have:

Theorem 4.1 (Hardness). *Problem 4.1 is* NP-COMPLETE.

We consider an alternative formulation for the above balanced partitioning problem. Note that while this reformulated problem is still NP-COMPLETE, as we see below, it is more convenient to design heuristic algorithms for it.

Problem 4.2 (Modified Merging). *Let $d_{max} = max_u\, d^u$, $u \in V_P$ be the maximum partition weight in the partition graph $G_P = (V_P, E_P)$. Split G_P into the smallest number of disjoint, connected components such that for each component, the sum of its partition weights is at most $k \times d_{max}$.*

By setting $k \leq d^*/d_{max}$ in the above problem, we can find connected components whose prior counts are estimated to be at most d^*. Observe that here, although we do not start out with a segmentation tree, the partitions provided by the partitioning algorithm can be thought of as leaf nodes of a segmentation tree and our merged components form parents, or ancestors of the leaf nodes.

Each component produced via a solution of Problem 4.2 also corresponds to an actual image segment formed by merging its member partitions: if the prior counts are accurate, these image segments together comprise a minimal terminating frontier for some segmentation tree. While in practice, they need not necessarily form a minimal terminating frontier, or even a terminating frontier, we observe that they provide very good approximations for one.

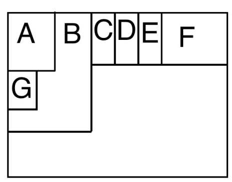


Figure 5: Articulation Point

Given the hardness of this modified merging problem, we now discuss good heuristics for it, and provide theoretical and experimental evidence in support of our algorithms.

FirstCut Algorithm. One simple approach to Problem 4.2, motivated by the first fit approximation to the Bin-Packing problem (Coffman Jr, Garey, and Johnson 1996), is to start a component with one partition, and incrementally add neighboring partitions one-by-one until no more partitions can be added without violating the upper bound, $k \times d_{max}$ on the sum of vertex weights. We refer to this as the `FirstCut` algorithm. In practice, however, we find that `FirstCut` performs suboptimally for several graphs as certain partitions and components get disconnected by the formation of other components during this process. For example, consider the partitioning shown in Figure 5.

Suppose partitions $A, \ldots, G$ contain 100 objects each and parameter $k = 6$. The maximum allowed size for a merged component is $6 \times d_{max} \geq 6 \times 100$. Supposing we start a component with A, and incrementally merge in partitions $B, \ldots, F$, we end up isolating G as an independent merged component. This causes some components to have fewer than $k \times d_{max}$ objects, which in turn will result in a higher final number of merged components than optimal.

ArticulationAvoidance Algorithm. Applying our first cut procedure to Figure 5 results in poor quality components if we merge partitions $B \ldots F$ to A before G. Intuitively, when adding B to the component containing A, the partition graph is split into two disconnected components: one containing G, and another containing $C \ldots F$. Given our constraint requiring connected components (contiguous images), this means that partition G can never be part of a reasonably sized component. This indicates that merging *articulation partitions* like B, i.e. , nodes or partitions whose removal from the partition graph splits the graph into disconnected components, potentially results in imbalanced final merged components. Since adding articulation partitions early results in the formation of disconnected components or imbalanced islands, we implement our `ArticulationAvoidance` algorithm that tries to merge them to growing components as late as possible. We merge partitions as before, growing one component at a time up to an upper bound size of $k \times d_{max}$, but we prioritize the adding of non-articulation partitions first. With each new partition, u, added to a growing component, we also update our list of articulation partitions for the new graph and repeat this process until all partitions have been merged into existing components.

We performed extensive evaluation of our algorithms on synthetic and real partition graphs and found that `ArticulationAvoidance` performs close to the theoretical optimum; `FirstCut`, on the other hand, often gets stuck at articulation partitions, unable to meet the theoretical optimum. For details of our algorithms, their complexities, and their evaluation on various partition graphs, we refer the reader to (Sarma et al. 2015).

4.3 Practical Setup

In this section we discuss some of the implementation details of and challenges faced by our algorithms in practice. Many of the challenges faced in Section 3.4 apply here as well.

Partitioning. The first step of our algorithm is to partition the image into small, non-overlapping partitions. To do this, we use the marker-controlled watershed algorithm (Beucher and Meyer 1992). The foreground markers are obtained by background subtraction using morphological opening (Beucher and Meyer 1992) with a circular disk.

Prior counts. In the example of Figure 4, we learn a model for the cells using a simple Support Vector Machine classifier. For a test image, every 15×15 pixel window in the image is classified as 'cell' or 'not cell' using the learned model – see (Sarma et al. 2015) for more details of this approach. Note that this procedure *always undercounts*, that is, the prior count estimate obtained for any partition is smaller than the true number of objects in that partition.

Traversing the Segmentation Tree. While Section 4.2 gives us a set of merged components, we still need to show these images to human workers to verify the counts. One option is to have (multiple) workers simply count each of these image components and aggregate the counts to get an estimate for the whole image. Since some of these image components may have counts higher than our set worker threshold of d^*, our model tells us that worker answers on the larger components could be inaccurate. So, another option is to use these images as a starting point for an expansion down the segmentation tree, and perform a `FrontierSeeking` search similar to that in Section 3 by splitting these segments until we reach segments whose counts are all under d^*. We compare these two alternatives in (Sarma et al. 2015) and find that while splitting the merged components could be beneficial for certain datasets, just our `AA` algorithm with worker counts on generated components is sufficient for our biological dataset.

5 Experimental Study

We deployed our crowdsourcing solution for counting on two image datasets that are representative of the many applications of our work. We examine the following questions:

- How do the JellyBean algorithms compare with the theoretically best possible "oracle" algorithms on cost?
- How accurate are the JellyBean algorithms relative to machine learning baselines?
- What are the monetary costs of our algorithms, and how do they scale with the number of objects?
- How accurate are the JellyBean algorithms relative to directly asking workers to count on the entire image?

5.1 Datasets

Dataset Description. Our first dataset is a collection of 12 images from Flickr. These images depict people in various settings, with the number of people (counts) ranging from 41 to 209. This is a challenging dataset, with people looking very different across images—ranging from partially to completely visible, and with varying backgrounds. Furthermore, no priors or partitions are available for these images—so we evaluate our solutions from Section 3 on this dataset. We refer to this as the *crowd dataset*.

The second dataset consists of 20 simulated images showing biological cells, generated using SIMCEP (Lehmussola et al. 2007). The number of objects in the images ranges from 151 to 328. The computer vision techniques detailed in Section 4 are applied on these images to get prior counts and partitions. We refer to this as the *biological dataset*.

Segmentation Tree. For the crowd dataset, the segmentation tree was constructed with fanout $b = 2$ until a depth of 5, for a total of 31 nodes per image. At each stage, the image was split into two equal halves along the longer dimension. This ensures that the aspect ratios of all segments are close to the aspect ratio of the original image. Given a segment, workers were asked to count the number of 'majority' heads (as described in Section 3.4)—if a head crossed the image boundary, it was to be counted only if the worker felt that majority of it was visible. To aid the worker in judging whether a majority of an object lies within the image, the surrounding region was shown demarcated by clear lines.

For the biological dataset, bins were generated by our `ArticulationAvoidance` algorithm.

Task Generation. The segments/bins, generated as above, were organized randomly into Mechanical Turk HITs (Human Intelligence Tasks) having 15 images each. The workers were paid 30¢ for each HIT. Across both datasets, workers provided counts for 2250 segments. Each HIT was answered by 5 workers and then take the `median` of their responses as the `WorkerCount`. We discuss additional experiments on worker behavior, as well as ones evaluating various answer aggregation schemes beyond `median` in (Sarma et al. 2015).

Given the generated segmentation trees, as well as the outcomes of the generated tasks, we are able to simulate the runs of different algorithms on the two datasets and compare them on an an equal footing.

5.2 Variants of algorithms

Algorithms for Both Datasets. For the above datasets, we evaluate the following algorithms:

- `FS`: our `FrontierSeeking` algorithm from Section 3;
- `OnlyRoot`: This algorithm queries only the root node of the segmentation tree, to test how workers perform without any algorithmic decomposition;
- `ML`: Machine learning baselines — (a) For the biological dataset, the prior counts from our machine learning algorithm from Section 4.3, and (b) For the crowd dataset, a pre-trained face detector from (Zhu and Ramanan 2012);
- `Optimal`: Given our worker behavior model, a worker's answer is expected to be accurate only if the number of objects to be counted is $< d^*$. Thus, any algorithm requires at least $\lceil \frac{\texttt{TrueCount}}{d^*} \rceil$ questions to count accurately, even if it knows the exact nodes to query. We call this `Optimal` since it is a lower bound for any algorithm given our error model.

Algorithms for Biological Dataset. For the biological dataset, we also evaluate the following algorithm:

- `AA`: `ArticulationAvoidance` algorithm (Section 4.1);

Ground Truth. For both datasets, we denote the true counts of images by `Exact`. While the (generated) images in the biological dataset have a known ground truth, the images in the crowd dataset were evaluated independently and agreed upon by two annotators.

Accuracy. The error of our algorithms is calculated as: $\frac{|\texttt{TrueCount-WorkerCount}|}{\texttt{TrueCount}}$. The percentage accuracy is therefore $100 \times (1 - \texttt{Error})$. We also use the percentage of images where $\texttt{TrueCount} = \texttt{WorkerCount}$ as another accuracy metric for the biological dataset.

5.3 Results

In this section, we describe the results of our algorithms.

> ***How do the JellyBean algorithms (`FS` and `AA`) compare with the theoretically optimal oracle algorithm on cost?***
> *On both datasets, the costs of `FS` and `AA` are within a small constant factor–between 1 to 2.5–of `Optimal`.*

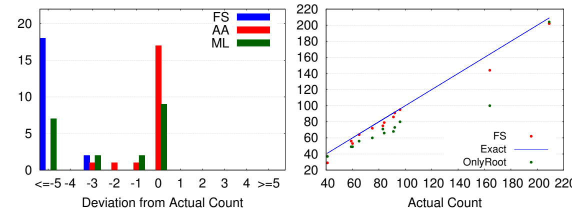
Crowd Dataset Optimality. For the crowd dataset, we compare the performance of `FS` against `Optimal`. Averaging across images, the number of questions asked by `FS` is within $2.3\times$ of `Optimal`. Further, this factor does not cross 2.75 for any image in the dataset. This is especially low considering how hard the images in this dataset are.

Biological Dataset Optimality. For the biological dataset, we compare the performance of `FS` and `AA` against `Optimal`. The average number of questions asked by `AA` is within a factor 1.35 of `Optimal`, which is significantly lower than the 2.3 factor for FS. This indicates that leveraging information from computer vision algorithms helps bring `AA` closer to "oracle" optimality.

> ***How accurate are the JellyBean algorithms (`FS` and `AA`) relative to machine learning baselines (`ML`)?***
> *On the crowd dataset, `FS` has a much higher accuracy of 97.5% relative to 70.1% for `ML` on the 5/12 images `ML` works on; for the remaining 7/12 images, `ML` detects no faces at all.*
> *On the biological dataset, `FS` has an accuracy of 96.4%. In comparison, `AA` increases the accuracy to 99.87%, returning exact counts for 85% of the images (off on the rest by counts of 1 to at most 3), while `Bio-ML` gets only 45% correct (off on the rest by counts of at least 5).*

Crowd Dataset Accuracy. For the crowd dataset, we compare the performance of `FS` against `ML`. On this difficult dataset, `ML` *fails to detect any faces for 7 out of the 12 images (i.e., making 100% error on 58.3% of the images), and has an average accuracy of 70.1% on the remaining*, demonstrating how challenging face detection can be to state-of-the-art vision algorithms. In comparison, `FS` counts people in all these images with an *average accuracy of 92.5% for an average cost of just $1.17 per image*. In particular, for the 5 images where `ML` did detect heads, the average accuracy of our algorithm was 97.5%. The accuracy of `FS`, which is independent of the domain, demonstrates that crowdsourcing can be very powerful for tasks such as counting.

Biological Dataset Accuracy. For the biological dataset, `FS` has an average accuracy of 96.4%. Next, we compare `AA` to `ML`, our computer vision algorithm, whose counts and partitions are input to `AA`. We observe that out of 20 images, `AA` *gets the correct* `Exact` *count for 17 (85%) of the images, while* `ML` *gets only 9 (45%) images exactly correct*. To study the errors further, we plot a histogram of the deviation from the correct counts in Figure 6a. The x-axis shows the deviation from `Exact` for an image, and the y-axis shows the frequency, or number of images for which the count estimated by an algorithm deviated for a specific x-value. We observe that even though the counts provided by `FS` are 96.4% accurate, they *deviate by more than 5 for 18/20 images*. `AA` is



(a) (b)

Figure 6: Accuracy: (a) Biological Dataset (b) Crowd Dataset

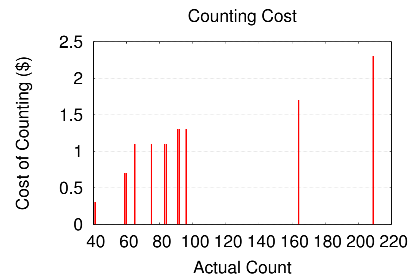


Figure 7: Crowd Dataset: Cost of counting

significantly better – *only 3 images deviating by counts of 1, 2, and 3 respectively. In comparison,* `ML` *estimate deviates by at least 5 for 7 images*. Thus, `AA`, which leverages both crowds and computer vision algorithms, outperforms both `FS` and `ML`.

> ***How expensive are the Jellybean algorithms?***
> *On both datasets with hundreds of objects, the algorithms `FS` and `AA` return accurate results at the cost of a few dollars per image. The cost of `AA` is approximately half of the cost of `FS` per object, indicating that "skipping ahead" in the segmentation tree using information from computer vision algorithms cleverly helps reduce cost significantly.*

Crowd Dataset Cost. In Figure 7 we plot the cost of counting an image from the crowd dataset using `FS` against the number of objects in that image. Each vertical slice corresponds to one image with its ground truth count along the x-axis, and dollar cost incurred along the y-axis. The cost is of the order of just a *few dollars even for very large counts*, making it a viable option for practitioners looking to count (or verify the counts of) objects in an image.

Biological Dataset Cost. The average cost of counting an image from the biological dataset incurred using `AA` was $1.6, as compared to $2.7 using `FS`. The average cost of counting per object was 0.63¢ for `AA` and 1.25¢ for `FS`. This significant reduction ($2\times$) for `AA` is a result of our merging algorithm which skips the larger granularity image segments and elicits accurate counts on the generated components.

How accurate are the JellyBean algorithms (`FS` and `AA`) relative to directly asking workers to count on the entire image (`OnlyRoot`)?
On the crowd dataset, `FS` *estimates counts with* $>$ 90% *accuracy on 9/12 images, relative to 2/12 images for* `OnlyRoot`.
On the biological dataset, `FS` *and* `AA` *improve the accuracy by 27% and 30.7% as compared to* `OnlyRoot`.

Crowd dataset. We now compare `OnlyRoot`, i.e., only asking questions at the root, versus `FS`. We plot the results in Figure 6b. The x-axis marks the ground truth (`Exact`) counts of images, while the y-axis plots the predicted counts by different algorithms. Each vertical slice corresponds to an image, and each point on the plot corresponds to the output of an algorithm for a given input image. We find that *average accuracy of* `OnlyRoot` *is 81.8% as compared to the 92.5% of* `FS`. We observe that splitting the image into smaller pieces improves counts significantly for most images. As Figure 6b shows, `FS` estimates better counts than `OnlyRoot` for 10/12 images. The two points on the extremes where `Onlyroot` yields a better answer result are anomalous due to image-specific reasons – see (Sarma et al. 2015).

Biological Dataset. For the biological dataset, the `OnlyRoot` baseline performs poorly, achieving an accuracy of $< 75\%$ for 14/20 images. In comparison, `FS` counts with an accuracy of 96.4% for all images. Further, `AA` has 100% accuracy on 17/20 images as shown in Figure 6a, indicating that using vanilla crowdsourcing without applying our JellyBean algorithms can lead to low accuracy.

6 Related Work

The general problem of finding, identifying, or counting objects in images has been studied in machine learning, computer vision and crowdsourcing communities. We discuss recent related work from each of these areas and compare them against our approach.

Unsupervised learning. A number of recent solutions to object counting problems tackle the challenge in an unsupervised way, grouping neighboring pixels together on the basis of self-similarities (Ahuja and Todorovic 2007), or similarities in motion paths (Rabaud and Belongie 2006). However, unsupervised methods have limited accuracy, and the computer vision community has therefore considered supervised learning approaches. These fall into three categories:

Counting by detection. In this category of supervised algorithms, a object detector is used to localize each object instance in the image. Training data for these algorithms is typically in the form of images annotated by bounding boxes for each object. Once all objects have been located, counting them is trivial (Nattkemper et al. 2002). However, object detection is an unsolved problem in itself even though progress has been made in recent years (Everingham et al. 2014).

Counting by regression. Algorithms in this category learn a mapping from image properties like texture to the number of objects. This mapping is inferred using one of the large number of available regression algorithms in machine learning e.g., neural networks (Cho, Chow, and Leung 1999; Marana et al. 1997). For training, images are provided with corresponding object counts. In such methods , the mappings from local features to counts are global, that is, a single function's learned parameters are used to estimate counts for the entire image or video. This works well when crowd densities are uniform throughout the image – a limiting assumption that is largely violated in real life applications.

Counting by annotation. A third approach has been to train on images annotated with dots. Instead of bounding boxes, each object here is annotated with a dot. For instance, in (Lempitsky and Zisserman 2010), an image density function is estimated, whose integral over a region in the image gives the object count. Another recent work counts extremely dense crowds by leveraging the repetitive nature of such crowded images (Idrees et al. 2013).

A common theme across these methods is that they deliver accurate counts when their underlying assumptions are met but are not applicable in more challenging situations. This guides us to leverage the 'wisdom of the crowds' in counting heterogeneous objects, which may be severely *occluded* by objects in front of them.

Crowdsourcing for image analysis. The above considerations indicate the requirement of human input in the object counting pipeline. The idea of using human inputs for complex learning tasks has recently received attention; in (Cheng and Bernstein 2015), the authors present a hybrid crowd-machine classifier where crowds are involved in both feature extraction and learning. Although crowdsourcing has been extensively used on images for tasks like tagging (Qin et al. 2011), quality assessment (Ribeiro, Florencio, and Nascimento 2011) and content moderation (Ghosh, Kale, and McAfee 2011), the involvement of crowds in image analysis has been largely restricted to generating training data (Sorokin and Forsyth 2008; Lasecki et al. 2013).

In a recent study of crowdsourcing for malaria image analysis (Luengo-Oroz, Arranz, and Frean 2012), non-expert players achieved a counting accuracy of more than 99%. In our work, we build on this study to propose solutions to the challenges that arise when using crowds to estimate counts in images across different application settings.

Summary. While there have been many studies on computer vision for counting and segmentation, either (a) the described settings are stylized or make application-specific limiting assumptions, or (b) the designed algorithms have relatively low accuracy in practice. Compared to the computer vision algorithms described, our approach to count objects is generic—it can be used to count heterogeneous, occluded objects in diverse images.

7 Conclusions

We tackle the challenging problem of counting the number of objects in images, a ubiquitous, fundamental problem in computer vision. While humans and computer vision algorithms, separately, are highly error-prone, our JellyBean algorithms combine the best of their capabilities to deliver high accuracy results at relatively low costs for two separate regimes or modes, while additionally providing optimality guarantees under reasonable assumptions.

Our JellyBean algorithms were shown to (a) be within a $2.75\times$ factor of the best possible oracle algorithm in terms of cost when operating without computer vision , and within a $1.3\times$ factor of the best possible oracle algorithm, with average cost reduced by almost half, when operating in concert with computer vision, (b) have high accuracy relative to both computer vision baselines as well as vanilla crowdsourcing.

Acknowledgements: We thank the anonymous reviewers for their valuable feedback. We acknowledge the support from grants IIS-1513407, IIS-1422977 and IIS-1453582 awarded by the National Science Foundation, grant 1U54GM114838 awarded by NIGMS through funds provided by the trans-NIH Big Data to Knowledge (BD2K) initiative (www.bd2k.nih.gov), and funds from Google and Amazon.

References

Ahuja, N., and Todorovic, S. 2007. Extracting texels in 2.1 d natural textures. In *ICCV 2007*, 1–8. IEEE.

Beucher, S., and Meyer, F. 1992. The morphological approach to segmentation: the watershed transformation. *Optical Engineering-New York-Marcel Dekker Incorporated-* 34:433–433.

Borodin, A. 1998. *Online computation and competitive analysis*, volume 2.

Cheng, J., and Bernstein, M. S. 2015. Flock: Hybrid crowd-machine learning classifiers. In *CSCW '15*.

Cho, S.-Y.; Chow, T. W.; and Leung, C.-T. 1999. A neural-based crowd estimation by hybrid global learning algorithm. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on* 29(4):535–541.

Coffman Jr, E. G.; Garey, M. R.; and Johnson, D. S. 1996. Approximation algorithms for bin packing: a survey. In *Approximation algorithms for NP-hard problems*, 46–93. PWS Publishing Co.

Dyer, M., and Frieze, A. 1985. On the complexity of partitioning graphs into connected subgraphs. *Discrete Applied Mathematics* 10(2):139–153.

Everingham, M.; Eslami, S. A.; Van Gool, L.; Williams, C. K.; Winn, J.; and Zisserman, A. 2014. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision* 111(1):98–136.

Forsyth, D. A., and Ponce, J. 2003. *Computer Vision: A Modern Approach*.

Ghosh, A.; Kale, S.; and McAfee, P. 2011. Who moderates the moderators?: crowdsourcing abuse detection in user-generated content. In *EC, 2011*, 167–176. ACM.

Idrees, H.; Saleemi, I.; Seibert, C.; and Shah, M. 2013. Multi-source multi-scale counting in extremely dense crowd images. In *CVPR*. IEEE.

Karger, D. R.; Oh, S.; and Shah, D. 2011. Iterative learning for reliable crowdsourcing systems. In *NIPS*, 1953–1961.

Lasecki, W. S.; Song, Y. C.; Kautz, H.; and Bigham, J. P. 2013. Real-time crowd labeling for deployable activity recognition. In *CSCW, 2013*. ACM.

Lehmussola, A.; Ruusuvuori, P.; Selinummi, J.; Huttunen, H.; and Yli-Harja, O. 2007. Computational framework for simulating fluorescence microscope images with cell populations. *Medical Imaging, IEEE Transactions on* 26(7).

Lempitsky, V., and Zisserman, A. 2010. Learning to count objects in images. In *Advances in Neural Information Processing Systems*, 1324–1332.

Liu, X.; Tu, P. H.; Rittscher, J.; Perera, A.; and Krahnstoever, N. 2005. Detecting and counting people in surveillance applications. In *AVSS, 2005*, 306–311. IEEE.

Loukas, C. G.; Wilson, G. D.; Vojnovic, B.; and Linney, A. 2003. An image analysis-based approach for automated counting of cancer cell nuclei in tissue sections. *Cytometry part A* 55(1):30–42.

Luengo-Oroz, M. A.; Arranz, A.; and Frean, J. 2012. Crowdsourcing malaria parasite quantification: an online game for analyzing images of infected thick blood smears. *Journal of medical Internet research* 14(6).

Marana, A.; Velastin, S.; Costa, L.; and Lotufo, R. 1997. Estimation of crowd density using image processing. In *Image Processing for Security Applications (Digest No.: 1997/074), IEE Colloquium on*, 11–1. IET.

Nattkemper, T. W.; Wersing, H.; Schubert, W.; and Ritter, H. 2002. A neural network architecture for automatic segmentation of fluorescence micrographs. *Neurocomputing* 48(1):357–367.

NBC News. 2005. Exhibit traces the history of the voting rights act. In *nbcnews.com/id/8839169/ns/us_news-life/t/exhibit-traces-history-voting-rights-act*.

Parameswaran, A. G.; Garcia-Molina, H.; Park, H.; Polyzotis, N.; Ramesh, A.; and Widom, J. 2012. Crowdscreen: Algorithms for filtering data with humans. In *SIGMOD, 2012*, 361–372. ACM.

Qin, C.; Bao, X.; Roy Choudhury, R.; and Nelakuditi, S. 2011. Tagsense: a smartphone-based approach to automatic image tagging. In *MobiSys*, 1–14. ACM.

Rabaud, V., and Belongie, S. 2006. Counting crowded moving objects. In *CVPR, 2006*, volume 1, 705–711. IEEE.

Ribeiro, F.; Florencio, D.; and Nascimento, V. 2011. Crowdsourcing subjective image quality evaluation. In *ICIP*, 3097–3100. IEEE.

Russell, J.; Couturier, S.; Sopuck, L.; and Ovaska, K. 1996. Post-calving photo-census of the rivière george caribou herd in july 1993. *Rangifer* 16(4):319–330.

Sarma, A. D.; Jain, A.; Nandi, A.; Parameswaran, A.; and Widom, J. 2015. Surpassing humans and computers with JELLYBEAN: Crowd-vision-hybrid counting algorithms. Technical report, Stanford University.

Sheng, V. S.; Provost, F.; and Ipeirotis, P. G. 2008. Get another label? improving data quality and data mining using multiple, noisy labelers. In *Proceedings of the 14th ACM SIGKDD*, 614–622. ACM.

Sorokin, A., and Forsyth, D. 2008. Utility data annotation with amazon mechanical turk. In *CVPR Workshops*.

Szeliski, R. 2010. *Computer vision: Algorithms and Applications*. Springer Science & Business Media.

Zhu, X., and Ramanan, D. 2012. Face detection, pose estimation, and landmark localization in the wild. In *CVPR, 2012*, 2879–2886. IEEE.