MadEye: Boosting Live Video Analytics Accuracy with Adaptive Camera Configurations

Mike Wong and Murali Ramanujam, *Princeton University;*
Guha Balakrishnan, *Rice University;* Ravi Netravali, *Princeton University*

https://www.usenix.org/conference/nsdi24/presentation/wong

This paper is included in the
Proceedings of the 21st USENIX Symposium on
Networked Systems Design and Implementation.

April 16–18, 2024 • Santa Clara, CA, USA

978-1-939133-39-7

Open access to the Proceedings of the
21st USENIX Symposium on Networked
Systems Design and Implementation
is sponsored by

MadEye: Boosting Live Video Analytics Accuracy with Adaptive Camera Configurations

Mike Wong[♠] Murali Ramanujam[♠] Guha Balakrishnan[◇] Ravi Netravali[♠]

[♠]*Princeton University* [◇]*Rice University*

Abstract

Camera orientations (i.e., rotation and zoom) govern the content that a camera captures in a given scene, which in turn heavily influences the accuracy of live video analytics pipelines. However, existing analytics approaches leave this crucial adaptation knob untouched, instead opting to only alter the way that captured images from fixed orientations are encoded, streamed, and analyzed. We present MadEye, a camera-server system that automatically and continually adapts orientations to maximize accuracy for the workload and resource constraints at hand. To realize this using commodity pan-tilt-zoom (PTZ) cameras, MadEye embeds (1) a search algorithm that rapidly explores the massive space of orientations to identify a fruitful subset at each time, and (2) a novel knowledge distillation strategy to efficiently (with only camera resources) select the ones that maximize workload accuracy. Experiments on diverse workloads show that MadEye boosts accuracy by 2.9-25.7% for the same resource usage, or achieves the same accuracy with 2-3.7× lower resource costs.

1 Introduction

Building on the steady growth in camera deployments and advances in deep neural networks (DNNs) for vision tasks (e.g., classification or detection) [8, 21, 46, 66, 70], live video analytics pipelines have become prevalent. These pipelines operate by continually streaming live video feeds from cameras to processing servers (either edge [9, 11, 76, 82, 109] or cloud [33, 56, 67, 119]), where DNNs are run on incoming frames to produce low latency and highly accurate results for different application queries, i.e., combinations of task, DNN, and object(s) of interest. Key use cases include autonomous driving, footfall tracking, traffic coordination, business analytics, among others [6, 10, 12, 25, 27, 31, 41, 42, 90, 91].

Given their practical importance, much research has been devoted to improving both the resource efficiency and accuracy of live video analytics pipelines. Existing solutions include accuracy-aware tuning of inference configuration, encoding, or appearance knobs [34, 40, 57, 85, 119], filtering out redundant content [26, 33, 53, 67], using cheaper model variants [9, 92], improving job scheduling [82, 96, 119], and so on. However, all of these works assume that the content observable by cameras is unchangeable, and instead can only be encoded, streamed, or analyzed differently. In essence, they focus on optimizing *fixed, preset* camera deployments.

Unfortunately, the deployment of cameras for analytics is itself a daunting task for operators. Subject to practical constraints (e.g., mounts, power sources), for a scene of interest, operators must determine the number of cameras to deploy and the orientation (i.e., combination of rotation and zoom factor) to use for each. There exist many possible orientations, and altering these decisions requires manual intervention. Yet we find that doing so can be highly fruitful: across different workloads and scenes, dynamically adapting orientations over time can yield accuracy improvements of 21.3-35.3% (without inflating resource usage) compared to even the *best* fixed-orientation scheme. These wins cannot be reaped by simply deploying more fixed cameras to simultaneously cover more orientations: most orientations are 'best' for short total times (median of 6 sec per 10-min video), drastically hindering the efficiency of such an approach, especially in resource-constrained settings where video analytics run [9, 68, 76, 97].

An alternative strategy is to leverage PTZ (pan-tilt-zoom) cameras that offer software libraries for tuning orientations, thereby providing a logical approach to capturing the above wins. Indeed, despite existing for nearly two decades, PTZ camera popularity has surged in recent years (global market value of $1.5 billion in 2021 that is expected to reach $3.6 billion by 2028 [20]) largely due to declining price points that can rival fixed-camera costs [1–4, 94, 102]. However, multiple challenges complicate their use for live analytics (§2.3). First, queries are highly sensitive, in different ways, to orientation knobs due to their diverse goals (e.g., tasks), inherent model biases (how models perceive scenes and objects), and scene dynamism (where objects are located) – optimizing orientation tuning for one workload can forego up to 25.1% of the potential median accuracy wins for another. Second, the 'grid' of orientations is large, but the selection space is sparse, with steep accuracy drops from the best orientation(s) to others at any time. Third, the best orientation changes rapidly, e.g., 85% of changes occur in ≤1 sec since the last change.

To overcome these issues, we present **MadEye**, a camera-server system that automatically and continually adapts PTZ camera orientations to maximize analytics accuracy for the scene and workload at hand. The key insight behind MadEye is that the speed at which commodity PTZ cameras can

change orientations (i.e., upwards of 600° per sec with concurrent zoom) far outpaces the rate at which applications require analytics results (typically 1-30 frames per second (fps), i.e., every 33-1000 ms). This, in turn, allows MadEye to eschew typical non-stationary multi-armed bandit strategies [64, 81, 107] that rely purely on previous explorations to determine orientation importance, in favor of a more informed strategy based on *current* scene content. Concretely, in each timestep (33 ms for 30 fps) and subject to network/compute resource availability, MadEye cameras explore multiple orientations and quickly determine which will maximize workload accuracy and warrant transmission to backend servers for full inference. Though intuitive, realizing this strategy in practice involves addressing several core challenges.

First, to enable fast camera-side evaluation of the importance of different orientations, MadEye adopts a custom knowledge distillation [49] strategy with edge-grade, ultra-compressed NN models. To cope with their potentially limited predictive power, we task them with modeling query sensitivities only to the point of accurately *ranking* orientations in terms of impact on workload accuracy – precise results are left to backend servers. Even with this relaxed framing, MadEye must employ several optimizations to achieve sufficient rank accuracy. Most notably, MadEye trains edge models using a common abstraction – detection for objects of interest – that reflects the minimum information needed to capture sensitivities and biases for popular tasks. Task-specific semantics need not be baked into edge models, and instead can be incorporated by post processing the generated detections.

Edge models are *continually* trained on MadEye's backend using both the latest and historical workload results to mitigate data skew towards recently-selected orientations. Importantly, to balance resource costs and accuracy, each edge model covers only a single query but all orientations. The intuition is that, while results from different query models can substantially divergen [14, 35, 63, 77], feature-level variance between orientations for the same scene is considerably narrower, often smaller than that in typical pre-training datasets [69]. Accordingly, MadEye freezes pre-trained feature extraction layers across queries, caching those weights on cameras to lower retraining and (downlink) model update overheads.

Second, we devise a novel, on-camera search strategy to explore orientations with the goal of capturing the best one (accuracy-wise) at each timestep. Three key empirical observations guide our search: (1) despite rapid temporal shifts, transitions between best orientations move slowly in the spatial dimension, (2) the best orientations at any time are usually clustered spatially, and (3) neighboring orientations (with overlapping views) have highly correlated trends in efficacy.

Building on these observations, MadEye explores a flexible shape of contiguous orientations at each timestep, and considers shifting only towards neighboring orientations whose efficacy can be robustly predicted. Decisions to keep/remove orientations are governed by both response rates (and the corresponding time budgets) and *relative* comparisons of recent edge model results. For the former, MadEye uses an efficient heuristic to determine path feasibility in the time budget (a variant of the NP-Hard Traveling Salesman Problem [48]). For the latter, MadEye gracefully trades off exploration (i.e., shape size) for network usage (i.e., sending more orientations for backend inference) to bound the effects of edge model errors and maximize accuracy for the required response rate.

To evaluate MadEye, we developed the first (to our knowledge) dataset that supports tuning rotation and zoom at each time instant by splicing out scenes of interest from publicly available 360° videos. Using this dataset, we evaluated MadEye on a variety of network conditions, commodity PTZ camera hardware, and workloads that incorporate multiple vision DNNs and query tasks: classification, counting (per-frame and aggregate), and detection. Across these settings, MadEye boosts accuracy by 2.9-25.7% compared to an oracle fixed-orientation strategy without inflating resource usage; these wins are within 1.8-13.9% of the oracle dynamic strategy. Framed differently, MadEye achieves those accuracy boosts with 2-3.7× lower resource footprints than the best strategy of using (multiple) fixed-orientation cameras. Moreover, MadEye outperforms recent PTZ tracking algorithms [93, 98] (by 2.0-3.8×) and multi-armed bandit solutions [106] (by 5.8×). The source code and datasets for MadEye are available at `https://github.com/michaeldwong/madeye`.

2 Background and Motivation

We start with an overview of live video analytics deployments (§2.1). We then show measurements highlighting the importance of dynamically adapting camera orientations to workloads and scenes (§2.2), and the challenges associated with realizing those benefits in practice (§2.3).

2.1 Overview of Live Video Analytics

In a live video analytics deployment, one or more cameras continually stream their video frames to servers for processing. Servers can range from distant (but powerful) cloud machines [96, 119] to nearby (but weaker) edge boxes [9, 76, 82], and are tasked with running queries on the incoming frames to support different applications. Queries most often involve running deep neural network (DNN) inference on individual frames, with the goals of locating and characterizing various objects in the scene, e.g., an intersection. Moreover, the queries for different applications can vary in terms of the tasks they perform, the objects they consider, the DNNs they use (different architectures and weights), and the response rates they require. For instance, footfall tracking for business analytics will count people passing through an area, with response rates at 1 fps or less [12]. In contrast, smart driving or sports analytics applications will detect the specific locations of cars or people, with response rates upwards of 30 fps [91].

In this paper, we focus on the following four query tasks (and their corresponding accuracy metrics) that have been prevalent in recent literature [23, 33, 58, 59, 67] and real-world

deployments [74, 82]; §3.4 details our target queries. We note that these tasks also serve as the building blocks for more complex queries, e.g., tracking relies on detections.

- **Binary classification**: asks if any objects of interest are present in a frame. Accuracy across the video is measured as the fraction of frames with the correct binary decision.
- **Counting**: counts the number of objects of interest in each frame. Accuracy for each frame is measured as the percent difference between the returned and ground truth counts.
- **Detection**: finds the precise bounding box coordinates for objects of interest in a frame. Accuracy per frame is measured using mAP [38], which evaluates the overlap between each returned box and its ground truth counterpart.
- **Aggregate counting**: counts the *unique* objects of interest that appear in a scene. Accuracy per video is the percent difference between the returned and ground truth counts.

Over time, an analytics deployment will face diverse workloads to run on the feeds it manages, each varying in query composition and size [77, 82]. Yet, the overarching goals persist: subject to resource constraints, deliver low-latency results (at the desired response rate) with maximal accuracy.

2.2 Opportunities with Tuning Camera Orientations

Existing video analytics systems (§6) assume that a stationary camera's orientation (rotation and zoom), and thus what it ingests from the target scene, is fixed and incapable of being adapted. To quantify the significance of this restriction, we run experiments on our 50-video dataset and workloads that incorporate 4 model architectures, the tasks from §2.1, and people/cars; §5.1 details our setup. Each video supports tuning of rotations (150° horizontally by 30°, 75° vertically by 15°) and zoom (1-3×); we use other granularities in §5.4.

For each video, we obtained per-frame (15 fps here) results for each workload by running its queries on all 75 orientations. We then define accuracy relative to the *best* orientation for each frame, i.e., the orientation that maximized per-frame accuracy for the workload. In other words, best orientations represent an upper bound on the accuracy that could be achieved by using a single orientation at a given timestep. For instance, for counting, an orientation's accuracy at any time is its object of interest count divided by the max count across all orientations at that time. Using this methodology, we compare three schemes: (1) *one time fixed* which selects the best orientation at time=0 and keeps it throughout the video, (2) *best fixed* which uses oracle knowledge to pick the best single orientation that maximizes average workload accuracy for the video, and (3) *best dynamic* which selects the best orientation per frame in the video.

As shown in Figure 1, adapting camera orientations brings substantial accuracy improvements without inflating resource usage, i.e., the same number of frames are transmitted and processed: median boosts with *best dynamic* are 30.4-46.3% over *one time fixed* and 21.3-35.3% over the *best fixed* scheme that

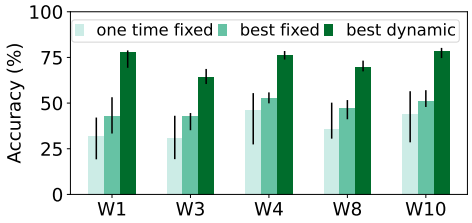


Figure 1: Accuracy for 5 representative workloads when using varying degrees of orientation adaptation. Bars list results for the median video, with error bars spanning 25-75th percentiles.

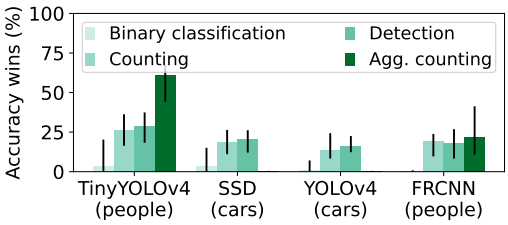


Figure 2: Accuracy wins from adapting orientations (compared to *best fixed*) grow as query specificity grows. Bars list median videos, with error bars for 25-75th percentiles. We exclude agg. counting+cars due to limits of multi-object trackers (§5.1).

is an upper bound for any fixed-orientation approach. Figure 2 breaks down these results by query task. Notably, the importance of adapting orientations grows as query types become more specific. For instance, for YOLOv4 and cars, median accuracy improvements over *best fixed* are 1.2%, 13.4%, and 16.4% for binary classification, counting, and detection. The reason is that coarser queries mask certain differences across orientations, e.g., if many objects of interest are present in the scene, any orientation that catches a single object will deliver max accuracy for binary classification; counting, on the other hand, will favor the orientation with the most objects.

Primer on PTZ cameras. Pan-tilt-zoom (PTZ) cameras present an intuitive mechanism to realize such adaptation. PTZ cameras come in two forms, traditional [36, 86] and electronic (ePTZ) [51, 87], both of which support software tuning of pan (horizontal rotation), tilt (vertical rotation), and zoom. The key difference between the two variants is in their tuning mechanisms. Traditional PTZ cameras embed physical motors to rotate at up to 600°-per-second and concurrently optically zoom (i.e., without reducing resolutions). In contrast, ePTZ cameras capture wide field-of-views and employ near-instantaneous digital rotation and zoom to focus on specific parts of the scene. ePTZ cameras change orientations faster and are cheaper, but also cover smaller rotation areas (150° vs. 360°) and degrade image quality by using digital zoom. PTZ cameras rival traditional ones in on-board compute resources, with recent offerings housing edge-grade GPUs [79].

2.3 Challenges

Despite the potential benefits of adapting camera orientations using PTZ cameras, three fundamental challenges complicate this approach in practice. We describe them in turn.

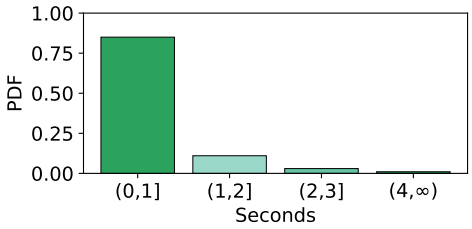


Figure 3: Shifts in the best orientation are frequent. Results list a PDF (binned by 1 sec) of time between switches in best orientation across all videos and workloads.

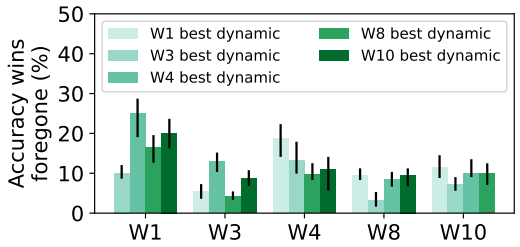


Figure 4: Workloads exhibit different sensitivity to orientations. Results apply the best orientations for workload X (legend) to workload Y (x axis), and plot the accuracy wins (over best fixed for Y) that are lost from not using the best orientations for workload Y. Bars list medians; error bars for 25-75th percentiles.

C1: rapid changes in best orientation over time. As shown in Figure 3, switches in best orientation are frequent: 85% of switches occur in ≤1 sec since the last switch. The reason for this high flux is threefold: (1) the typical motion for certain objects (e.g., moving car or walking person) is sufficient to frequently cross orientation boundaries, especially since orientations exhibit a degree of content overlap, e.g., an object may be visible in two orientations and quickly move to change the one it is most prominent in, (2) models can provide inconsistent results across even back-to-back frames that are seemingly unchanged [60, 61], so best orientation swaps can occur even with static objects, and (3) aggregate queries seek previously unseen objects and thus intentionally move to new orientations, e.g., the percentage of sub-second switches mildly drops to 70% when aggregate queries are excluded.

C2: diverse workload sensitivities to zoom and rotation. At any point in time, the best orientation can vary across individual queries and workloads. Figure 4 illustrates this, showing that adapting orientations to maximize accuracy for one workload can result in foregoing 3.2-25.1% of the potential (median) accuracy wins for other workloads.

Figure 5 highlights this at a query level, showing that different models, objects, and tasks can all influence orientation selections. Model discrepancies influence what can be discerned in the scene during inference and under what orientations. For instance, with people counting, selecting best orientations for a YOLOv4 query will miss out on 26.3% median accuracy wins for the same task using SSD (even when trained on the same dataset). In contrast, tasks dictate the specificity needed in the collected results, e.g., optimizing for counting people

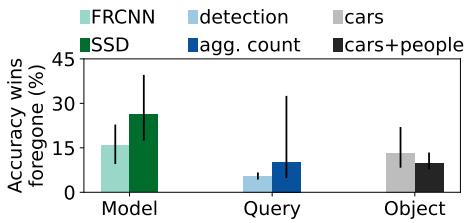


Figure 5: Applying the best orientations for a base query of {YOLOv4, counting, people} to a query Y that modifies a single element in the base query; we compare the accuracy wins (over best fixed) to those when using the best orientations for Y. Bars list medians; error bars for 25-75th percentiles.

rather than aggregate people counting with the same model foregoes 10.2% of potential wins. Lastly, object type governs the importance of regions based on object densities, as well as the features used for and difficulty in detecting relevant objects (smaller objects are tougher to discern [88]). Thus, optimizing for a YOLOv4 people counting query would forego 13.3% of wins if the query considered cars instead.

Figure 6 provides example screenshots to illustrate the benefits and harm of changing orientations. Importantly, tuning orientations does not simply bring new objects into field of view, and instead *plays a large role in a model's ability to detect objects that were already visible*. Indeed, simply using the lowest zoom factor will capture the largest portion of the scene, but can result in models missing in-view objects due to degradations in object size or resolution.

C3: massive (but sparse) search space. The orientation space exhibits substantial sparsity in the spatial and temporal dimensions. For the former, among the 75 orientations at any time, only 1 (or several, with ties) is best, with steady dropoff in accuracy to the others, e.g., median dips of 4.8% and 20.7% from the best to 2nd and 5th best. For the latter, most orientations are best for short total times in each video, with median durations of 5-6 sec across workloads (Figure 7).

3 Design

Figure 8 shows the end-to-end operation of MadEye. The main insight behind MadEye is to leverage fast PTZ rotation speeds to explore many orientations in each timestep (i.e., between when results are needed for an fps), and then select, based on their *current* content, the one(s) that maximize workload accuracy under resource constraints. The idea is to limit the "guess work" compared to prior search algorithms that rely only on past orientation efficacy (§5.3).

As in other video analytics systems [33, 57, 67, 82], users register queries with a backend agent (on an edge or cloud server), specifying a target scene, as well as a model to use, object(s) of interest, and a task. To operate under camera compute constraints, MadEye then trains edge-compatible (i.e., highly compressed) models (§3.1), not to replace the original (more accurate) query models (as in typical knowledge distillation [49]), but instead to *approximately* extract information of importance in a frame for each query. In other words, ap-

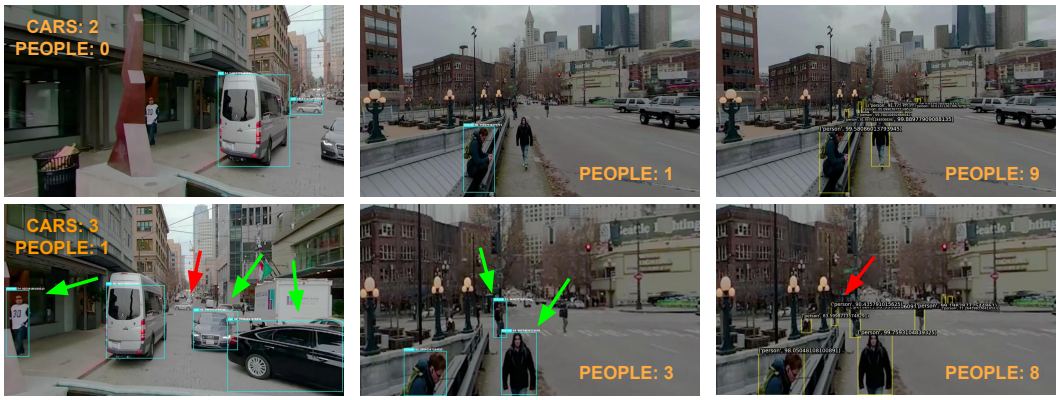


Figure 6: Screenshots showing the (diverse) impact of rotation and zoom for different queries. Each column shows two images from the same time instant that use either different rotation or zoom. On the bottom row, green arrows show newly captured objects, while red arrows show objects that are newly missed after the orientation change. Left: rotation brings a new object into the scene, helps detect 2 previously-visible objects, but loses a previously-detected object. Middle: zooming in helps detect new people. Right: after switching models, the same zoom from the middle column actually reduces the number of detected people.

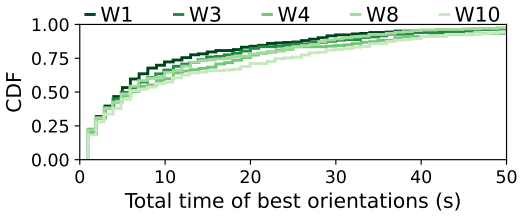


Figure 7: Most orientations are best for short total times in each video. Results consider all orientation-video pairs per workload.

proximation models are explicitly designed to estimate the inherent sensitivities of each query (C2 from §2.3).

To cope with the large space of orientations and rapid shifts in best ones (C1 and C3 from §2.3), MadEye employs an efficient on-camera search strategy (§3.3) that explores as many potentially fruitful orientations as possible while avoiding fps violations for results. The camera then runs approximation models on all captured orientations in each timestep and uses the results to (1) *rank* the orientations in terms of their likelihood to maximize workload accuracy, and (2) determine the orientations to consider in the next timestep. The highest ranked orientations that the network can support are sent to the backend for full workload inference, with results used to continually adapt approximation models to the scene (§3.2).

3.1 Designing Approximation Models

The primary objective of MadEye's approximation models is to quantify the *relative* importance of orientations for the queries in a workload. However, this requires capturing the sensitivity of each query to different orientation and scene dynamics, subject to camera compute constraints. Given the potential complexity of workload queries, we eschew noisy (and limited) vision features based on local gradients [30, 73] in favor of knowledge distillation with compressed models [49]. However, we alter this approach in several ways to favorably balance ranking accuracy and resource efficiency.

We design approximation models using a common abstraction that reflects the minimum amount of information needed to sufficiently rank orientations. The key idea is that the core elements of query sensitivity pertain to how models find and characterize objects, rather than how tasks post-process those results. Thus, MadEye's approximation models are structured purely as ultra-lightweight detectors for objects of interest; this strategy also avoids tricky development of compressed models per task. Concretely, we use the smallest variant of the edge EfficientDet family [103], EfficientDet-D0 (3.9M parameters, >150 fps on a Jetson edge GPU), which enables MadEye to scale to multi-query workloads. More complex detectors could be used, but cameras possess limited GPU memory [67, 82], and inference delays negatively influence the degree to which MadEye can explore orientations (§3.3).

Why a detector? Two alternatives we considered for the approximation models are to directly estimate object counts in an image, and to directly output rank orderings across multiple images. However, we empirically observed high error rates with both. This is largely because such approaches can only relate the presence of features to objects via a global regression over an entire image (or multiple images), failing to leverage local regressions via bounding box predictions to boost precision. While image-level DNN object counters do exist [99, 113, 118, 122], they focus on large crowds of people. In contrast, there are often few objects of interest in an orientation at any time (§2.3), making rank orderings extremely sensitive to small errors in count prediction.

MadEye uses one approximation model per query, rather than per workload or object. Though more efficient, we avoid per-workload and per-object approximation models as we (like others [77]) find that different DNNs can exhibit wildly varying response profiles to even the same object classes due to object-independent factors like scale and resolution [50].

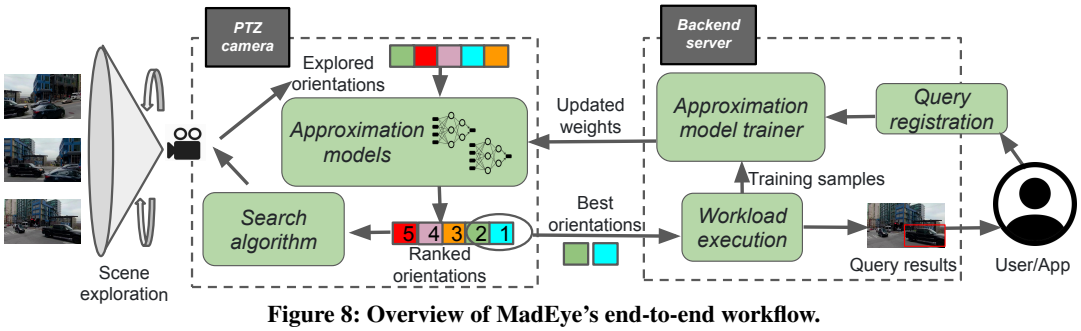


Figure 8: Overview of MadEye's end-to-end workflow.

Moreover, DNNs trained on very different datasets are known to inherit different algorithmic biases [14, 35, 63, 80, 100, 111].

However, each approximation model is configured to support *all* orientations for two reasons. First, the number of orientations is large (§2.3), making per-orientation approximation models impractical with on-camera GPUs. Second, neighboring orientations exhibit substantial overlap, and since we only consider orientations for a given scene, divergence in background content, lighting, shadows, etc. are minimal. Indeed, we measured the perceptual distance [89] of images (LPIPS) from different orientations in the same scene to be 0.30. For context, the same value for the popular MS-COCO and Pascal VOC datasets used to successfully pre-train many vision models (including EfficientDet) are 0.46 and 0.41.

Estimating workload accuracies. MadEye post-processes the generated bounding boxes from all approximation models to compute *predicted workload accuracies* for orientation ranking. To do this, MadEye follows the per-task accuracy metrics from §2.1, but computes per-orientation predicted accuracy in a relative manner compared to the other orientations under test. For instance, counting computes the ratio of object counts between each orientation and the max among the set of explored orientations at that timestep, while detection expands this to incorporate object area sizes (as per mAP score). Lastly, aggregate counting modulates count scores to favor less explored orientations (that may have unseen objects).

3.2 Continually Training Approximation Models

MadEye servers train a new approximation model for each new query, with the goals of being fast and accurate (in ranking orientations). Initial training uses a small set of 1000 historical images from the target scene that is then labeled (online) using the DNN in the registered query; label generation takes 7-90 sec depending on the DNN. However, to accelerate this process, MadEye begins with a version of EfficientDet that is pre-trained on Pascal VOC, and freezes both the backbone and BiFPN layers responsible for feature extraction and fusion. Only weights for the final 3 bounding box and class prediction layers are fine-tuned to mimic the target query's behavior. The rationale is that model features progressively move from general (e.g., textures, gradients) to task-specific (e.g., object labels) from model start to end [16, 116, 117]. Initial fine-tuning lasts 40 epochs (≈25 mins).

Even after initial fine-tuning, approximation models may fail to generalize to changing scene dynamics [101], leading to degrading accuracy. To cope with such data drift, MadEye employs continual learning (every 120 sec) to update the model's weights using the latest query results on orientations sent to the server for full workload inference. While continual learning has been applied to edge video analytics [9, 75], MadEye requires several alterations from prior efforts. The main challenge is that within each retraining window, samples are only available for the orientations that MadEye's camera-side component recently visited and deemed worthy of backend inference. Since orientations are typically best for short total times (§2.3), there is often severe imbalance in the orientations covered by new training samples. For instance, with perfect rankings, the average 2-minute window sees only 9.3% of orientations get sent to the backend. This can result in overfitting to certain orientations, and catastrophic forgetting [62] for others that may soon be ranked highly.

To deal with this, MadEye retrieves the most recent historical training samples from each orientation and uses this to balance the dataset. As we will discuss in §3.3, we find that orientation shifts are often spatially localized, with changes to distant orientations happening over longer timescales. Thus, MadEye pads the data samples for neighboring orientations (up to 3 away from the latest one) to match the count for the most popular orientation in the retraining window. The remaining orientations use an exponentially declining number of samples based on their distance from the latest orientation.

Each continual learning round considers the last 120 best orientations selected by MadEye (1 sample per second since the last retraining round), as well as 200 historical images (on average) to account for neighboring orientations; a random 30% of this is reserved for validation. Retraining runs for 5 epochs and takes an average of 32 seconds. Note that all continual learning runs asynchronously on backend servers; §5.4 profiles network usage for shipping model updates which are small due to the use of ultra-lightweight (compressed) detectors and the aforementioned backbone freezing.

3.3 Exploring and Ranking Orientations

The primary goal of MadEye's on-camera component is to efficiently explore (a subset of) the large orientation space to capture the best orientation for each timestep. Realizing

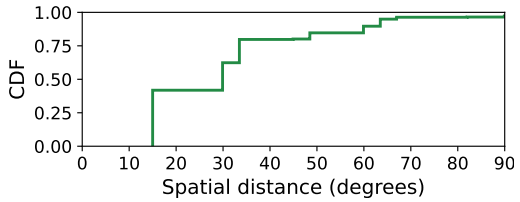


Figure 9: Spatial distance between successive best orientations is small, with most transitions between neighboring orientations. Results aggregate across all videos and workloads for 15 fps.

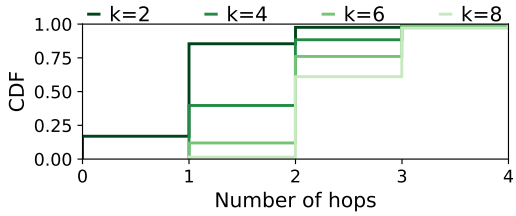


Figure 10: Top ranked orientations are often spatially clustered. Results use 15 fps, are aggregated across all workloads and videos, and show the max distance between orientations in the top k ranked orientations at each timestep.

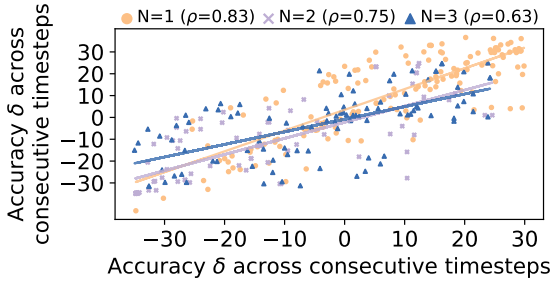


Figure 11: Correlation in accuracy changes across orientations separated by N hops. Results list Pearson Correlation Coefficients and cover 3 representative videos and workloads (15 fps).

this is challenging for three reasons. First, MadEye only has visibility into the orientations that it has recently explored, but other orientations can change in content and importance at any time. Second, even among recently explored orientations, MadEye only has access to coarse results from approximation models (i.e., that accurately capture only relative importance) for most. Third, each timestep is not only dedicated to exploration, but also (1) running approximation models on explored orientations, (2) encoding and shipping select orientations to the server, and (3) running the workload on shipped images.

Rather than relying on previous (and potentially stale) observations at each orientation (§5.3), MadEye opts for a more informed strategy guided by 3 empirical observations.

- Although best orientations change rapidly over time (§2.3), those changes are far slower in the spatial dimension. Figure 9 illustrates this, showing that the median and 90th percentile spatial distance between successive best orientations are 30° and 63.5°, which pertains to shifts spanning only 1 or 2 orientations in our default grid (§5.1).
- The best performing orientations (accuracy-wise) at any time are often spatially clustered (Figure 10). Concretely, across our dataset, the 75th percentile distance separating orientations in the top k at each timestep is 1 and 2 orientations for k values of 2 and 6.
- Accuracy for neighboring orientations often shift in tandem. Indeed, as shown in Figure 11, the correlation coefficient for accuracy changes in direct neighbors is 0.83; intuitively, this value shrinks to 0.75 when considering neighbors 2-hops away (that exhibit less content overlap).

Taken together, these findings motivate a search strategy that considers a flexible shape of contiguous orientations at each timestep, and swaps out underperforming orientations in the previous shape only for neighboring ones whose trends we can robustly predict for the next timestep. We start with a description of the algorithm that does not account for zoom or resource constraints and later incorporate those elements. Common themes are: only relative comparisons of approximation model results are used, we leverage all outputs from those models (including bounding boxes), and search decisions are entirely local (i.e., on cameras) to remain rapid.

MadEye begins with a rectangular seed shape that reflects the largest coverable area in the time budget, thereby maximizing early exploration; we reset to this shape any time 0 objects of interest are found in a shape. The corresponding orientations are captured and analyzed with approximation models to compute a predicted workload accuracy for each (§3.1). After sending the top k orientations to the server for workload inference, MadEye must use these prior results to determine the set of orientations to explore in the next timestep.

To do this, MadEye labels each orientation from the last timestep with a value that indicates the likelihood of being fruitful in the *next* timestep. Concretely, we combine the exponentially weighted moving averages from recent (10) timesteps for (1) any computed predicted accuracy values, and (2) the deltas between those values. Weighted averages are used to remain robust to inconsistencies in DNN results across consecutive frames [77, 84], which is especially pronounced with MadEye's compressed approximation models.

Using those labels, MadEye must now determine which orientations to remove and add for the upcoming timestep. For this, MadEye sorts orientations into an ordered list based on their label values. Using pointers at the head H (largest label) and tail T (smallest label) of the list, MadEye iteratively compares orientations by asking: should we remove the orientation at T in favor of adding a neighbor to H? Concretely, MadEye computes the ratio of label values for H/T. If (1) that ratio exceeds a threshold (indicating a substantial disparity in the potential of H and T), (2) H has neighbors not already in the shape, and (3) removing T would not break contiguity, we remove the orientation at T and increment the pointer. The process repeats by considering the addition of another neighbor for H, this time using a larger threshold to account for the additional uncertainty of adding more neighbors. H is decremented when a neighbor cannot be added, and the process ends when even one neighbor for H cannot be added.

For each iteration that results in a neighbor addition for H, MadEye selects among H's neighbors by analyzing the bounding boxes that its approximation models generated in the last timestep. For each candidate neighbor, we compute the ratio of two values: normal distances to the center of H and to the centroid of all bounding boxes in H. Values <1 indicate lower chances of H's objects moving to the candidate in the next timestep. We repeat this process for all other orientations in the last shape that the candidate exhibits any non-zero overlap with. Candidate neighbor scores are computed as the weighted sum of these ratios (weights according to degree of overlap), and the candidate with the max score is selected.

Reachability and path selection. The search algorithm thus far ignores whether a PTZ camera can sufficiently cover the selected shape in a given time budget. Formally, the shape of orientations can be represented as a fully-connected undirected graph with edge weights pertaining to the time taken to move between two adjacent orientations (given a rotation speed). Our goal is to determine whether the shape is coverable in a given time budget, and if so, what is the shortest path. The paths between orientations satisfy the triangle inequality property [104], so this can be modeled as a variant of the NP-Hard Traveling Salesman Problem (TSP) [18]. Given our tight time budgets, MadEye employs the Minimum Spanning Tree (MST) heuristic [48], but optimizes it to minimize online delays. In particular, since our orientation grid is static, we precompute pairwise distances and the entire MST ahead of time. Online, for a given shape, we quickly extract and perform a preorder walk on the corresponding subgraph to get the shortest path. This reduces the heuristic to linear complexity (in orientations); each path computation takes 14 μs, and the resultant paths are within 92% of optimal. Upon failure, MadEye greedily removes the orientation with the lowest potential (that does not break contiguity) and rechecks reachability.

Balancing search size and network/compute delays. MadEye pipelines its exploration through orientations with the running of approximation models on each one. However, network transmission to and workload inference on the backend do not overlap with orientation exploration. The reason is that transmissions are governed by global ranks across *all* orientations explored in each timestep. Thus, in each timestep, we face a tradeoff between exploring more orientations and sending more orientations to the backend.

MadEye resolves this tension based on the expected difficulty for its approximation models to accurately rank the considered orientations, which in turn governs the risk associated with exploring more orientations (and sending fewer to get ground truth results). Intuitively, scenarios where the considered orientations are projected to contribute similar accuracies pose the biggest difficulty for approximation models (as the gaps between ranks shrinks). MadEye determines the right balance by first selecting a target number of frames to send according to the training accuracy for approximation models (provided by the backend) and the variance in predicted accuracy values in the last timestep, e.g., with 85% training accuracy, any frames within 15% accuracy of the top ranked frame are sent. MadEye then computes a target shape size for exploration, accounting for network delays (harmonic mean of past 5 transfers [115]), backend compute delays, rotation speeds, and approximation model inference delays.

Handling zoom. After selecting the set of orientations to visit, the search algorithm must determine the zoom factor to use for each one. The challenge is that past accuracies are insufficient for determining zoom fidelity as MadEye cannot know what objects are being missed by not zooming in/out. Instead, we rely on bounding boxes from approximation models to determine the risk of zooming in. When an orientation is added to the shape, we start at the lowest zoom factor to gain visibility into its whole content. At each timestep, we compute the average distance between each bounding box and the centroid of all boxes; smaller distances indicate more clustering and less risk of zooming in. These values are compared with the area covered by each zoom factor to select one, and MadEye automatically zooms out after 3 seconds to avoid missing newly entering objects in the orientation.

Transmitting images. At the end of each timestep, MadEye must transmit select images to the server for workload inference. Unlike standard streaming, MadEye sends disjoint sets of images from each orientation's video stream. To keep bandwidth costs low, MadEye maintains a list of the last image shared for each orientation, and employs a functional encoder [39] that computes deltas relative to that image.

3.4 Query Support and Deployment Discussion

As noted in §2.1, MadEye targets the large class of object-centric queries (i.e., those that find and label people or cars) which have dominated reported industrial workloads [74, 82] and prior work [23, 33, 58, 59, 67, 75, 77]. Further, we consider tasks that return per-frame (e.g., counting, detection) and per-video (e.g., aggregate counting) results.

Supporting other object-centric queries with new objects of interest or tasks involves two main steps: training a new approximation model, and (potentially) developing new logic to rank orientations using approximation model results. The former is automated; MadEye does not make any assumptions about object type or task, and instead trains its approximation models as detectors for the objects of interest identified by results of provided query models. The latter is only required for new tasks and can be arbitrarily complex, e.g., a ranker can consider keypoints of detected objects for activity recognition queries. §A.1 presents results for other object types and tasks.

MadEye currently operates on videos from cameras that are *statically mounted*. For changing mounting locations (e.g., dashcams), MadEye will face similar challenges to any multi-armed bandit strategy: past content becomes a weaker indicator of orientation efficacy since orientations continually point

to new scenes. MadEye may still outperform prior approaches by comparing explored orientations in terms of current content, but we leave a detailed exploration to future work.

4 Implementation

MadEye's core components are written in 9.1k lines of Python code, with all training and inference tasks across the backend and camera run in PyTorch. We use TensorRT [7] to accelerate inference on the backend, and a variant of Nexus [96] as a round-robin scheduler for approximation model inference. Orientations are first represented as rotational values, projected onto a 360° space, and then converted using an in-house equirectangular-to-rectilinear image converter (in C++) to match the APIs offered by PTZ cameras [17]. For ground truth accuracy computations (§5.1) that need a global perspective on object locations and uniqueness, atop the ByteTrack multi-object tracker [114] that links objects across an orientation's video, we use `cv2` and `scikit-image` to extract image features (e.g., SIFT) that link objects across orientations.

5 Evaluation

We evaluated MadEye across diverse workloads, network settings, and videos. Our key findings are:

- MadEye increases median workload accuracies by 2.9-25.7% compared to an oracle fixed-orientation strategy (while using the same amount of resources); wins are within 1.8-13.9% of the oracle dynamic strategy.
- Achieving MadEye's accuracy wins with 1 PTZ camera would require the best 4-6 fixed-orientation cameras, which comes with a 2-3.7× inflation in resource costs.
- MadEye outperforms prior PTZ algorithms by 2.0-5.8×, providing 31.1-52.7% higher accuracy than Panoptes [98], tracking [93], and multi-armed bandits [106].
- MadEye gracefully balances on-camera exploration and transmission of orientations to maximize accuracy even as resources shrink and response rates rise.

5.1 Methodology

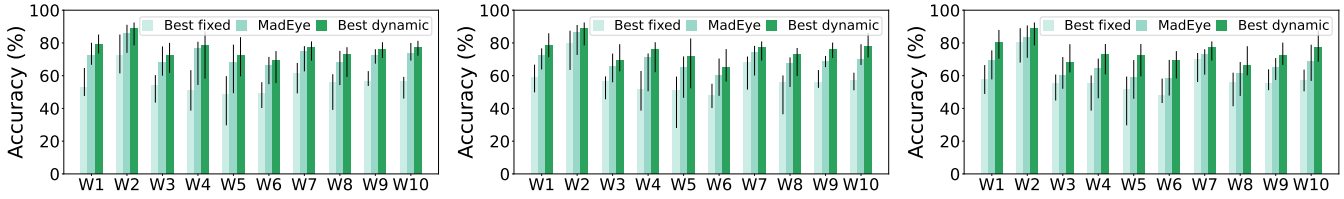
Video dataset for PTZ analysis. To the best of our knowledge, there does not exist a public video dataset for PTZ cameras that enables users to tune rotation and zoom knobs; instead, existing PTZ datasets reflect pre-determined knob decisions. Thus, to evaluate MadEye, we generate our own dataset. To construct our dataset, we begin with the abundance of 360° datasets. Concretely, we use 50 360-degree videos (5-10 mins each) from YouTube that incorporate scenes of interest resembling those from prior video analytics work [9, 33, 67], e.g., searching for people and/or cars in traffic intersections, walkways, shopping centers. From each video, we carve out the scenes of interest as regions that each span 150° horizontally and 75° vertically. We then subdivide each scene into grids of orientations to mimic recent PTZ offerings [51] (30° and 15° granularities for pan and tilt; we explore other grids in §5.4), and extract a full video per orientation. For zoom, since we operate on pre-captured videos, we employ digital zoom (1-3×) by cropping images and scaling back the dimensions to match the original image.

Models and workloads. We consider 4 popular architectures for vision tasks: SSD [71] and Faster RCNN [95] with ResNet-50 backbones, YOLOv4 and Tiny-YOLOv4 [108] with CSPDarknet53 backbones. We consider two versions of each model trained on Pascal VOC and MS-COCO, but show results for the latter as the trends were similar. To construct queries, we follow the same methodology from production analyses [82]. Each model can perform any of the 4 tasks from §2.1 with a focus on people or cars. We enumerate all possible workloads sized between 2-20 queries and pick 10 randomly. §A.2 details each workload. As in prior work [23, 82], we run each workload on all videos in our corpus that contain the objects of interest for the workload's queries. Since MadEye does not make any assumptions about frame arrival rates, we consider response rates between 1-30 fps.

Hardware and networks. On-camera computations run on an edge-grade Jetson Nano [79] equipped with a 128-core Maxwell GPU, quad-core ARM CPU with 1.43 GHz clock speed, and 4 GB of memory. We consider default camera rotation speeds of 400° per second and study this parameter in §5.4. Workload inference and training of approximation models run on a server with an NVIDIA RTX 2080 Ti GPU (8 GB RAM) and 18-core Intel Xeon 5220 CPU (2.2 GHz; 125 GB RAM). Camera and server components are connected with emulated Mahimahi networks [78] using fixed-capacity (24-60 Mbps; 5-20 ms) and real-world mobile traces.
Note. Although we run on real edge hardware and emulated networks, we do not use a deployed PTZ camera for our main evaluations as it would preclude practical consideration of diverse scenes and camera parameters, e.g., different rotation speeds. We present results with a real PTZ camera in §5.5.

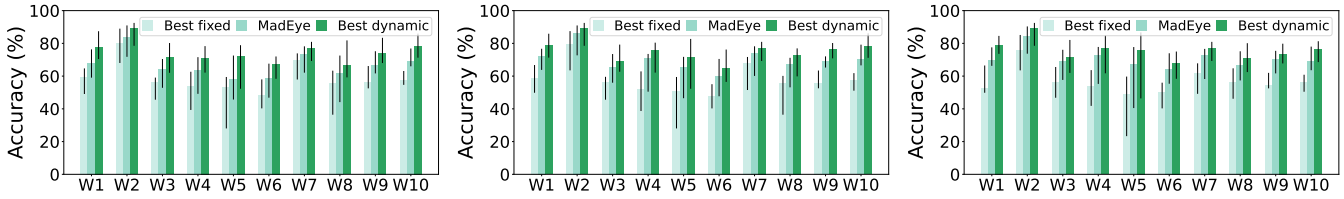
Metrics. Our primary evaluation metric is average workload accuracy per video. For each frame, following the accuracy definitions from §2.1, we compute per-orientation accuracy for each query relative to the orientation that delivers the max accuracy at that time. Per-query accuracies at each time are averaged to compute per-frame workload accuracies, which in turn are averaged to compute workload accuracy for a video.

While computing these values for binary classification and counting are straightforward, detections and aggregate counting need slight alterations. For detections, mAP scores depend on bounding box coordinates for specific objects and thus cannot be measured by comparing results across orientations. Thus, we consolidate the bounding boxes across orientations into a global view, and employ de-duplication [83] to eliminate redundant objects in overlapping regions. We then compute each orientation's mAP score relative to the global scene, and assign per-orientation accuracies as the ratio of its mAP



(a) 1 fps (b) 15 fps (c) 30 fps

Figure 12: Comparing MadEye with the best possible fixed- and adaptive-orientation schemes across all videos and workloads with a {24 Mbps, 20 ms} network and varying fps. Bars list medians with errors bars spanning 25-75th percentiles.



(a) Verizon LTE (b) 24 Mbps; 20 ms (c) 60 Mbps; 5 ms

Figure 13: Comparing MadEye with the best possible fixed- and adaptive-orientation schemes across all videos and workloads with fixed fps (15) and varying networks (improving from left to right). Bars list medians with errors bars spanning 25-75th percentiles.

score to the max one across orientations. For aggregate counting which is evaluated across an entire video, we compute the ratio of unique objects across the orientations that a system selects with the total number of unique objects in the video. Note that ByteTrack (§4) was unable to robustly support car tracking, so we exclude aggregate counting for cars.

5.2 Overall Results

We first compare MadEye with the two baselines from §2.2, *best fixed* and *best dynamic*, on different network and fps settings. Both baselines impractically rely on oracle knowledge of video content and workload accuracy, i.e., to pick the best orientation per video or per timestep, respectively, that maximizes accuracy for the target workload-video. Nonetheless, they serve as useful context for MadEye's performance. Note that MadEye automatically adapts the number of frames it explores and transmits based on network delays and response rates (§3.3). For *best fixed*, we leverage increasing network speeds by adding more fixed cameras (i.e., best, 2nd best, etc.), rather than simply capturing more (redundant) frames from 1 camera. *Best dynamic* does not change for any query other than aggregate counting, for which we send the largest number of fruitful orientations that the network can support.

Our results are captured in Figures 12-13. Across these settings, MadEye delivers median and 75th percentile accuracies that are 2.9-25.7% and 1.6-20.7% higher than *best fixed*, and within 1.8-13.9% and 1.3-12.5% of *best dynamic*. Digging deeper, our results show two key trends. First, as frame rates decrease (for a fixed network), MadEye's accuracies and wins over *best fixed* grow, e.g., for a {24 Mbps, 20 ms} network, median wins improve from 5.8-13.3% to 12.4-25.7% as fps drops from 15 to 1. The reason is that lower fps yields larger timesteps (e.g., 1 sec for 1 fps, 66.7 ms for 15 fps), enabling more exploration and/or transmission. Second, as network speeds grow (for fixed fps), the same trends persist (since each network transfer is faster) but to a lesser extent, e.g., median 15 fps wins grow to 8.6-18.4% for {60 Mbps, 5 ms}.

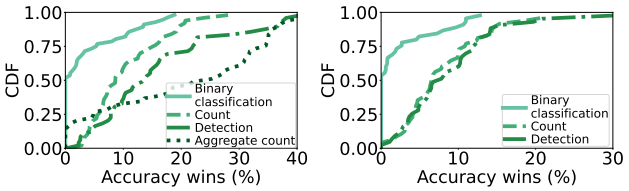


Figure 14: MadEye's accuracy wins (over best fixed) for different tasks and objects (left: people, right: cars). Results consider all videos and models, and use 15 fps and {24 Mbps; 20 ms}.

MadEye Variant	Median Accuracy (%)	# Fixed Cameras
MadEye-1	63.1	3.7
MadEye-2	66.3	5.5
MadEye-3	66.8	6.1

Table 1: Number of optimally-configured fixed cameras needed to match the accuracy of MadEye. MadEye-*k* refers to a version of MadEye that is restricted to sending the top *k* frames to the server for workload inference. Results consider a {24 Mbps; 20 ms} network, 15 fps response rate, and all video-workload pairs.

Figure 14 breaks down MadEye's wins over *best fixed* by task and object. Following the rationale from §2.2, accuracy boosts with MadEye grow as task specificity grows: median wins grow from 8.6% to 13.3% to 22.1% as we move from counting to detections to aggregate counting for people. We also observe consistently larger accuracy wins for people queries (rather than cars) due to their less structured motion patterns (more frequent and scattered orientation switches), e.g., for detections, wins for cars shrink to 6.7%.

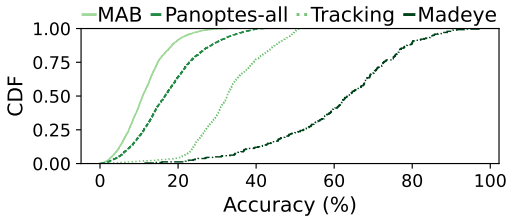


Figure 15: MadEye vs. 3 camera tuning strategies. Results are for all workloads and videos, 15 fps, and {24 Mbps; 20 ms}.

Results thus far focus on accuracy improvements. However, a key goal with MadEye is to maximize accuracy for a given resource cost, i.e., network and backend inference overheads. Table 1 lists the smallest number of optimally configured fixed cameras that would be required to match the accuracies that different versions of MadEye deliver, each of which sends a different number of frames per timestep. As shown, it would take 3.7 fixed cameras to realize the 63.1% accuracy that MadEye-1 achieves, implying a 3.7× reduction in network and backend compute usage. MadEye-2 is matched by 5.5 fixed cameras; here, however, the resource reduction factor is 2.8× since MadEye also sends 2 frames per timestep.

5.3 Comparisons with State-of-the-Art

We compare MadEye with 3 alternate approaches for adaptive camera orientations. Figure 15 shows results for a {24 Mbps; 20 ms} network and 15 fps; trends hold for all other scenarios.

First, we consider Panoptes [98], a recent PTZ system that configures orientations for workloads of applications, each explicitly concerned with specific orientation(s). For orientations of relevance, Panoptes generates a static round-robin schedule that is weighted according to how many queries an orientation is of interest to and how much motion has been detected historically in that orientation; higher weights indicate staying in an orientation for longer. Panoptes then switches between orientations according to this schedule with one exception: if motion gradients in the direction of any overlapping orientation of interest exceed a threshold, Panoptes switches there for several sec before resuming the round robin. Panoptes does not specify a zoom strategy, so we consider the best zoom (accuracy-wise) for any orientation it visits.

We consider two versions of Panoptes, *Panoptes-all* and *Panoptes-few*, in which each workload query is interested in all orientations or only its best fixed orientation, respectively. Max accuracy in both cases is defined relative to the best orientation among only the set of considered ones. As shown in Figure 15, MadEye outperforms Panoptes-all by 3.8×, with 46.8% higher accuracy at the median. The reason is that Panoptes cycles through orientations based on a pre-determined schedule and motion gradients *in the current orientation*, neither of which are sufficient indicators of importance of other orientations at the current time, e.g., orientations are suboptimal most of the time (§2.3). In contrast, MadEye considers many orientations per timestep, ranking them based on current content. The wins persist compared to Panoptes-few (not shown due to the different accuracy metric), but are less pronounced (median of 40.5%) as there are fewer unfruitful orientations for Panoptes to consider.

System	Resource reduction	Median accuracy
Chameleon [57]	2.4×	46.3%
Chameleon + MadEye	2.4×	56.1%

Table 2: MadEye preserves resource savings of recent systems, while improving accuracy. Results use 15 fps, {24 Mbps; 20 ms}.

Next, we consider tracking algorithms that most PTZ cameras come equipped with today [93]. This algorithm starts in a home region (best fixed in our experiment), selects the largest object it finds, and tracks that object continually across orientations aiming to keep it as centered as possible. The algorithm resets to the home region upon losing the tracked object. We consider a favorable variant in which all orientations explored in a timestep are shared with the backend, which uses the one with the highest accuracy. As shown, MadEye delivers 2.0× higher workload accuracies (31.1% more at the median) compared to this tracking scheme. The main reason again is that the presence of a large object is a poor indicator of accuracy importance as it fails to capture more general scene properties and specific query sensitivies.

Finally, we consider the common UCB1 multi-armed bandit (MAB) algorithm [106]. Each orientation is considered a lever with a weight set to the average observed accuracy across all past visits (we seed this with historical data). The algorithm continually selects an orientation to visit as the one with the highest sum of weighted average and upper confidence bound (which favors less-visited orientations). As with tracking, we send all visited orientations to the backend, which selects the best one per timestep. MadEye delivers 52.7% higher median accuracies than this scheme, i.e., a 5.8× win. Unlike the schemes above, MAB does factor in workload accuracies in selecting orientations. However, its adaptation considers only historical efficacy (not current content), and scene dynamics have shifted by the time it updates its patterns.

Compatibility with other optimizations. By focusing on previously un-tuned knobs (rotation and zoom) to boost accuracy, MadEye is largely compatible with prior efforts that optimize resource overheads. To illustrate this, we consider a variant of Chameleon [57] that dynamically tunes pipeline knobs (resolution and frame rate) to lower network and backend inference resource costs without harming accuracy; we brute force selections per frame focused on the best fixed orientation. We then run MadEye atop the fps and resolution selections that Chameleon makes, sending the same amount of network data. As shown in Table 2, Chameleon lowers resource costs by 2.4× compared to the naive scheme that sends all frames at the highest resolution; MadEye preserves these efficiency wins, while increasing accuracy by 9.8%.

5.4 Deep Dive Results

Rotation speeds. We evaluated the impact of camera rotation speed on MadEye's performance by considering values of

{200, 400, 500, infinite}° per second, a fixed network ({24 Mbps; 20 ms}), and 15 fps. Intuitively, accuracy grows as rotation speeds increase, e.g., jumping from 54.2% to 64.9% as rotation speed grows from 200 to 500° per second. The reason is that faster rotations enable the exploration of additional orientations or, in rarer instances, additional transmissions. Importantly, benefits plateau since most queries (other than aggregate counting) are fully satisfied accuracy-wise as long as MadEye finds the best orientation at each timestep.

Grid granularity. To understand the effect of grid granularity (with other settings fixed), we focus on the pan dimension (since it is wider) and consider steps of {15, 30, 45, 60}°. Overall, MadEye's accuracy benefits shrink as grids become more fine-grained (with more orientations), e.g., median accuracies drop from 67.5% to 51.8% when pan steps drop from 45 to 15. This is because, although exploration in a time budget is governed by rotation speeds rather than grid granularity, the same distance (in °) of exploration will warrant more approximation model inference on more orientations, thereby shrinking each timestep's exploration budget.

Overheads. On MadEye's backend, the primary overheads are in initializing approximation models and continually sharing model updates with the camera. Across our workloads, we find median bootstrapping delays to be 27 mins (including labeling and initial fine-tuning). Downlink streaming consumes 3.2 Mbps for the median experiment. Recall that both overheads are mitigated by MadEye's fine-tuning strategy (§3.2). On cameras, the main overheads are in selecting orientations to explore and running approximation models; for the median workload-video pair, per-timestep delays for each task were 17 μs and 6.7 ms for 15 fps and {24 Mbps; 20 ms}. The former benefits from pre-computed reachability analysis (§3.3).

Microbenchmarks. MadEye's performance is governed by two main tasks: (1) ranking orientations with approximation models, and (2) selecting orientations to explore to find the best one(s) per timestep. For the former, Figure 16 (in §A) show that MadEye's approximation models assign median ranks of 1.1-1.3 to the best explored orientation at each timestep, significantly outperforming the variant that relies on counting directly on images. For the latter, for the median workload-video pair on {24 Mbps; 20 ms} and 15 fps, MadEye explores best orientation 89.3% of the time, with 6.8% of errors coming from our conservative zoom strategy (§3.3).

Downlink network speeds. MadEye servers periodically ship updated weights for approximation models to cameras for orientation ranking. To understand the impact that downlink network speeds have on MadEye, we augmented the set of networks used in our main experiments with two network scenarios that deliver far slower transmissions: Narrowband-IoT ({10 Mbps; 50 ms} on average) and AT&T 3G network ({2 Mbps; 100 ms} on average). Across five representative videos and workloads (at 15 fps) from our corpus, we observed increases in weight transmission times from {11, 5, 2} seconds for Verizon LTE, {24 Mbps; 20 ms}, and {60 Mbps; 5 ms} networks to {13, 66} seconds for the two new networks. These delays, in turn, resulted in mild accuracy degradations of up to 0.9% and 2.1% relative to MadEye running on the {24 Mbps; 20 ms} network. The reason is that on such short timescales, previous approximation models deliver only moderate errors in orientation ranking as they were trained just several minutes in the past. Those errors are further bound by the fact that MadEye's search strategy moves slowly spatially to ensure that the *several* top-ranked orientations are all considered for ranking at each timestep (§3.3).

5.5 On-Camera Evaluation

To demonstrate that commodity PTZ offerings (and their corresponding tuning APIs and rotational motors) can support the computational and exploratory requirements that MadEye imposes, we ran experiments using the PTZOptics PT12X-USB [5]. We selected a random set of 6 videos and 4 workloads from our main corpus, and considered response rates of 1-30 fps and the networks from §5.1. For each scenario, we fed the video into the PTZ camera at 30 fps. MadEye's camera-side processing ran on a Jetson Nano and interacted directly with the camera through its native HTTP interface for tuning orientations; the server was the same as in §5.1.

Overall, the camera was able to support the required exploration and computation load. However, we observed several rare artifacts that deviate from our main setup: (1) seemingly random, though minor, delays in API responsiveness for tuning orientations, and (2) small delays to reach max rotation speed, which our setup failed to capture by not modeling the physical mechanics of camera motors. These discrepancies mildly reduced the number of orientations that MadEye could explore in especially high-exploration windows with scattered orientations, but the effect on accuracy was minimal: across the experiments, wins over *best fixed* dropped by <1%.

Note that these results do not evaluate the long-term wearing effect of high exploration on PTZ camera components, which falls outside the scope of this paper and our setup. However, the results in §5.4 show that MadEye can still provide substantial accuracy benefits even with highly restricted rotation speeds (and thus, exploration rates).

6 Related Work

Adapting video analytics knobs. VideoStorm [120] selects an input knob configuration (e.g., frame rate, resolution) per workload to lower resource costs and facilitate job scheduling on servers. Chameleon [57] extends such tuning to adapt to dynamic scenes, while Llama [40] further tunes knobs across heterogeneous hardware to meet latency targets. By focusing on tuning camera orientations, MadEye provides complementary benefits to these efforts, boosting accuracies while preserving their resource efficiency wins (§5.3). Other efforts focus on camera-side knobs as MadEye does. For example, CamTuner [85] learns to boost accuracy by automatically

tuning capture knobs that cameras do not usually auto-adjust, e.g., brightness and contrast. AccMPEG [34] predicts the effects of macroblock encoding settings on DNNs, and tunes encoding to maximize accuracy. MadEye shares the goal of these efforts – tune camera knobs to boost workload accuracy – but focuses on complementary knobs, i.e., orientations.

Frame filtering and result reuse. Many prior efforts exploit temporal redundancies in video data by filtering out frames for network transfer and processing, and reusing results accordingly [13, 22, 26, 29, 32, 43, 44, 65, 67, 84, 112, 121, 124]. Spatula [53] extends this to multi-camera settings, selecting among cameras in a network. These optimizations are logically similar to MadEye, which also aims to maximize accuracy per network usage. However, the techniques are largely complementary: filtering decisions could be made among explored orientations to maximize new content in transfers.

Computation and network optimizations. Several efforts lower compute footprints either by identifying lightweight model variants [19, 28, 45, 49, 52, 72, 92, 123], sharing model layers during inference [55, 82], or using smarter job scheduling [96, 120]. Other systems lower network overheads by compressing transmitted frames in a manner that is recoverable on the server or does not negatively impact accuracy [33, 37, 110]. MadEye is entirely complementary to both directions in that it solely focuses on judiciously selecting images (i.e., orientations) to process at any time for an application-provided model (which can be compressed).

Drone coordination. Many efforts adapt drone flight plans (and thus what on-board cameras see) to maximize analytics accuracy or scene coverage [15, 47, 54, 105]. However, these systems focus on identifying events of interest (e.g., wildfires) in a geographically dispersed area for a preset application. In contrast, MadEye tunes camera orientations for a single scene to cope with workload nuances and maximize accuracy.

7 Conclusion

MadEye continually tunes PTZ camera orientations to maximize accuracy for a given analytics workload and resource setting. Key to MadEye are a rapid algorithm that searches through the large space of orientations at each time, and a new, approximate knowledge distillation strategy that efficiently selects the most fruitful (accuracy-wise) orientations from those explored. Across diverse workloads and settings, MadEye improves accuracy by 2.9-25.7% or resource costs by 2.0-3.7× without affecting the other.

Acknowledgements. We thank Wyatt Lloyd, Huacheng Yu, Kevin Hsieh, Olga Russakovsky, the NSDI reviewers, and our shepherd, Danyang Zhuo, for their helpful and constructive comments. This work was supported by a Sloan Research Fellowship and NSF CNS grants 2147909, 2151630, 2140552, 2153449, and 2152313.

References

[1] C289 2MP WiFi PTZ Security Camera + Person/Vehicle Detection + 128GB Local Storage. https://www.zositech.com/products/wifi-security-camera-pan-tilt-zoom-works-with-alexa-c289?variant=44502418063602.

[2] Logitech PTZ PRO 2 . https://www.logitech.com/en-us/products/video-conferencing/conference-cameras/ptz-pro2-conferencecam.960-001184.html?utm_source=google&srsltid=AfmBOorJvnI86m27PDAg-lycP4cFBE732NYvfwAHdZ4bGdnxovP1y6VAN5s.

[3] OTTICA NDI®|HX PTZ Video Camera 20x Optical Zoom POE 1080/60p (White) . https://ikancorp.com/shop/cameras/ptz-cameras/ottica-ndihx-ptz-video-camera-20x-optical-zoom-poe-1080-60p-white/.

[4] ReoLink 5MP Smart PTZ WiFi Indoor Camera. https://reolink.com/us/product/e1-zoom/.

[5] USB PTZ Cameras. https://ptzoptics.com/usb/.

[6] Video analytics traffic study creates baseline for change. https://www.govtech.com/analytics/Video-Analytics-Traffic-Study-Creates-Baseline-for-Change.html, 2020.

[7] NVIDIA TensorRT. https://developer.nvidia.com/tensorrt, April 2021.

[8] Pytorch-yolov3. https://github.com/eriklindernoren/PyTorch-YOLOv3, 2021.

[9] Ekya: Continuous learning of video analytics models on edge compute servers. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 119–135, Renton, WA, April 2022. USENIX Association.

[10] Video analytics market. https://www.fortunebusinessinsights.com/industry-reports/video-analytics-market-101114, 2022.

[11] Ganesh Ananthanarayanan, Victor Bahl, Landon Cox, Alex Crown, Shadi Nogbahi, and Yuanchao Shu. Video analytics - killer app for edge computing. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '19, pages 695–696, New York, NY, USA, 2019. Association for Computing Machinery.

[12] Larry Anderson. Video analytics applications in retail - beyond security. https://www.securityinformed.com/insights/co-2603-ga-co-2214-ga-co-1880-ga.16620.html/.

[13] Kittipat Apicharttrisorn, Xukan Ran, Jiasi Chen, Srikanth V. Krishnamurthy, and Amit K. Roy-Chowdhury. Frugal following: Power thrifty object detection and tracking for mobile augmented reality. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, SenSys '19, page 96–109, New York, NY, USA, 2019. Association for Computing Machinery.

[14] Guha Balakrishnan, Yuanjun Xiong, Wei Xia, and Pietro Perona. Towards causal benchmarking of biasin face analysis algorithms. In *Deep Learning-Based Face Analytics*, pages 327–359. Springer, 2021.

[15] Favyen Bastani, Songtao He, Ziwen Jiang, Osbert Bastani, and Sam Madden. Skyquery: An aerial drone video sensing platform. In *Proceedings of the 2021 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2021, page 56–67, New York, NY, USA, 2021. Association for Computing Machinery.

[16] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6541–6549, 2017.

[17] Francois Beaufort. Control camera pan, tilt, and zoom – Pan, tilt, and zoom features on cameras are finally controllable on the web. https://web.dev/camera-pan-tilt-zoom/.

[18] Michael A Bender and Chandra Chekuri. Performance guarantees for the tsp with a parameterized triangle inequality. In *Algorithms and Data Structures: 6th International Workshop, WADS'99 Vancouver, Canada, August 11–14, 1999 Proceedings 6*, pages 80–85. Springer, 1999.

[19] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033*, 2020.

[20] Business Research Insights. Global ptz camera market research report 2020. https://www.businessresearchinsights.com/market-reports/ptz-cameras-market-100130.

[21] Zhaowei Cai, Mohammad Saberian, and Nuno Vasconcelos. Learning complexity-aware cascades for deep pedestrian detection. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 3361–3369, Washington, DC, USA, 2015. IEEE Computer Society.

[22] Christopher Canel, Thomas Kim, Giulio Zhou, Conglong Li, Hyeontaek Lim, David G. Andersen, Michael Kaminsky, and Subramanya R. Dulloor. Scaling video analytics on constrained edge nodes. In *2nd SysML Conference*, 2019.

[23] Frank Cangialosi, Neil Agarwal, Venkat Arun, Junchen Jiang, Srinivas Narayana, Anand Sarwate, and Ravi Netravali. Privid: Practical, privacy-preserving video analytics queries. In *Proceedings of the 19th USENIX Conference on Networked Systems Design and Implementation*, NSDI'22, Berkeley, CA, USA, 2022. USENIX Association.

[24] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7291–7299, 2017.

[25] David Cassel. Are we ready for ai-powered security cameras? https://thenewstack.io/are-we-ready-for-ai-powered-security-cameras/.

[26] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 155–168, 2015.

[27] Mary Collins. The Hudl Algorithm: Turning Video into Player Tracking Data. https://www.maryecollins.com/hudl-tracking.

[28] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[29] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. Clipper: A Low-Latency online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 613–627, Boston, MA, March 2017. USENIX Association.

[30] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005.

[31] Sokemi Rene Emmanuel Datondji, Yohan Dupuis, Peggy Subirats, and Pascal Vasseur. A survey of vision-based traffic monitoring of road intersections. *Trans. Intell. Transport. Sys.*, 17(10):2681–2698, October 2016.

[32] Utsav Drolia, Katherine Guo, Jiaqi Tan, Rajeev Gandhi, and Priya Narasimhan. Cachier: Edge-caching for recognition applications. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 276–286, 2017.

[33] Kuntai Du, Ahsan Pervaiz, Xin Yuan, Aakanksha Chowdhery, Qizheng Zhang, Henry Hoffmann, and Junchen Jiang. Server-driven video streaming for deep learning inference. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 557–570, New York, NY, USA, 2020. Association for Computing Machinery.

[34] Kuntai Du, Qizheng Zhang, Anton Arapin, Haodong Wang, Zhengxu Xia, and Junchen Jiang. Accmpeg: Optimizing video encoding for accurate video analytics. In D. Marculescu, Y. Chi, and C. Wu, editors, *Proceedings of Machine Learning and Systems*, volume 4, pages 450–466, 2022.

[35] Mengnan Du, Fan Yang, Na Zou, and Xia Hu. Fairness in deep learning: A computational perspective. *IEEE Intelligent Systems*, 36(4):25–34, 2020.

[36] Jinlong E, Lin He, Zhenhua Li, and Yunhao Liu. Wisecam: Wisely tuning wireless pan-tilt cameras for cost-effective moving object tracking. In *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*. IEEE, 2023.

[37] John Emmons, Sadjad Fouladi, Ganesh Ananthanarayanan, Shivaram Venkataraman, Silvio Savarese, and Keith Winstein. Cracking open the dnn black-box: Video analytics with dnns across the camera-cloud boundary. In *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges*, HotEdgeVideo'19, page 27–32, New York, NY, USA, 2019. Association for Computing Machinery.

[38] Mark Everingham, Luc Gool, Christopher K. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *Int. J. Comput. Vision*, 88(2):303–338, June 2010.

[39] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S. Wahby, and Keith Winstein. Salsify: Low-latency network video through tighter integration between a video codec and a transport protocol. In *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation*, NSDI'18, page 267–282, USA, 2018. USENIX Association.

[40] Francisco Romero, Mark Zhao, Neeraja J. Yadwadkar, Christos Kozyrakis. Llama: A Heterogeneous & Serverless Framework for Auto-Tuning Video Analytics Pipelines. SoCC '21. ACM, 2021.

[41] Isha Ghodgaonkar, Subhankar Chakraborty, Vishnu Banna, Shane Allcroft, Mohammed Metwaly, Fischer Bordwell, Kohsuke Kimura, Xinxin Zhao, Abhinav Goel, Caleb Tung, et al. Analyzing worldwide social distancing through large-scale computer vision. *arXiv preprint arXiv:2008.12363*, 2020.

[42] Grand View Research. Global Sports Analytics Market Size Report, 2021-2028. https://www.grandviewresearch.com/industry-analysis/sports-analytics-market.

[43] Peizhen Guo, Bo Hu, Rui Li, and Wenjun Hu. Foggycache: Cross-device approximate computation reuse. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, MobiCom '18, page 19–34, New York, NY, USA, 2018. Association for Computing Machinery.

[44] Peizhen Guo and Wenjun Hu. Potluck: Cross-application approximate deduplication for computation-intensive mobile applications. *SIGPLAN Not.*, 53(2):271–284, mar 2018.

[45] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, page 123–136, New York, NY, USA, 2016. Association for Computing Machinery.

[46] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[47] Songtao He, Favyen Bastani, Arjun Balasingam, Karthik Gopalakrishna, Ziwen Jiang, Mohammad Alizadeh, Hari Balakrishnan, Michael Cafarella, Tim Kraska, and Sam Madden. Beecluster: Drone orchestration via predictive optimization. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, MobiSys '20, page 299–311, New York, NY, USA, 2020. Association for Computing Machinery.

[48] Michael Held and Richard M Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.

[49] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[50] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311, 2017.

[51] HuddleCamHD. Understanding the difference between EPTZ and PTZ . https://huddlecamhd.com/eptz-and-ptz/.

[52] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018.

[53] Samvit Jain, Xun Zhang, Yuhao Zhou, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Victor Bahl, and Joseph Gonzalez. Spatula: Efficient cross-camera video analytics on large camera networks. In *ACM/IEEE Symposium on Edge Computing (SEC 2020)*, November 2020.

[54] Sagar Jha, Youjie Li, Shadi Noghabi, Vaishnavi Ranganathan, Peeyush Kumar, Andrew Nelson, Michael Toelle, Sudipta Sinha, Ranveer Chandra, and Anirudh Badam. Visage: Enabling timely analytics for drone imagery. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, MobiCom '21, page 789–803, New York, NY, USA, 2021. Association for Computing Machinery.

[55] Angela H. Jiang, Daniel L.-K. Wong, Christopher Canel, Lilia Tang, Ishan Misra, Michael Kaminsky, Michael A. Kozuch, Padmanabhan Pillai, David G. Andersen, and Gregory R. Ganger. Mainstream: Dynamic stem-sharing for multi-tenant video processing. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 29–42, Boston, MA, July 2018. USENIX Association.

[56] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. Chameleon: Scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 253–266, New York, NY, USA, 2018. ACM.

[57] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. Chameleon: Scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 253–266, New York, NY, USA, 2018. Association for Computing Machinery.

[58] Daniel Kang, Peter Bailis, and Matei Zaharia. Blazeit: Optimizing declarative aggregation and limit queries for neural network-based video analytics. *Proc. VLDB Endow.*, 13(4):533–546, December 2019.

[59] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. Noscope: Optimizing neural network queries over video at scale. *Proc. VLDB Endow.*, 10(11):1586–1597, August 2017.

[60] Kai Kang, Hongsheng Li, Junjie Yan, Xingyu Zeng, Bin Yang, Tong Xiao, Cong Zhang, Zhe Wang, Ruohui Wang, Xiaogang Wang, and Wanli Ouyang. T-CNN: Tubelets With Convolutional Neural Networks for Object Detection From Videos. *IEEE Trans. Cir. and Sys. for Video Technol.*, 28(10):2896–2907, October 2018.

[61] Kai Kang, Wanli Ouyang, Hongsheng Li, and Xiaogang Wang. Object detection from video tubelets with convolutional neural networks. In *CVPR*, 2016.

[62] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018.

[63] Aditya Khosla, Tinghui Zhou, Tomasz Malisiewicz, Alexei A Efros, and Antonio Torralba. Undoing the damage of dataset bias. In *Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part I 12*, pages 158–171. Springer, 2012.

[64] Judy Goldsmith Kshitija Taywade, Brent Harrison. Using non-stationary bandits for learning in repeated cournot games with non-stationary demand. *arXiv preprint arXiv:2201.00486*, 2022.

[65] Adarsh Kumar, Arjun Balasubramanian, Shivaram Venkataraman, and Aditya Akella. Accelerating deep learning inference via freezing. In *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, Renton, WA, July 2019. USENIX Association.

[66] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua. A convolutional neural network cascade for face detection. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5325–5334, June 2015.

[67] Yuanqi Li, Arthi Padmanabhan, Pengzhan Zhao, Yufei Wang, Guoqing Harry Xu, and Ravi Netravali. Reducto: On-camera filtering for resource-efficient real-time video analytics. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 359–376, New York, NY, USA, 2020. Association for Computing Machinery.

[68] Zhuqi Li, Yuanchao Shu, Ganesh Ananthanarayanan, Longfei Shangguan, Kyle Jamieson, and Victor Bahl. Spider: A multi-hop millimeter-wave network for live video analytics. In *ACM/IEEE Symposium on Edge Computing*. ACM/IEEE, December 2021.

[69] Michael Lin, Tsung-Yiand Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *Computer Vision – ECCV 2014*, pages 740–755, 2014.

[70] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, July 2017.

[71] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In *Computer Vision – ECCV 2016*, pages 21–37. Springer International Publishing, 2016.

[72] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2736–2744, 2017.

[73] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60, 2004.

[74] Yao Lu, Aakanksha Chowdhery, and Srikanth Kandula. Optasia: A relational platform for efficient large-scale video analytics. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, SoCC '16, pages 57–70, New York, NY, USA, 2016. ACM.

[75] Mehrdad Khani, Ganesh Ananthanarayanan, Kevin Hsieh, Junchen Jiang, Ravi Netravali , Yuanchao Shu, Mohammad Alizadeh , Victor Bahl. Recl: Responsive resource-efficient continuous learning for video analytics. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, Boston, MA, April 2023. USENIX Association.

[76] Microsoft Azure. Azure stack edge, May 2021.

[77] Neil Agarwal, Ravi Netravali. Boggart: Towards General-Purpose Acceleration of Retrospective Video Analytics. NSDI '23. USENIX, 2023.

[78] R. Netravali, A. Sivaraman, K. Winstein, S. Das, A. Goyal, J. Mickens, and H. Balakrishnan. Mahimahi: Accurate Record-and-Replay for HTTP. Proceedings of ATC '15. USENIX, 2015.

[79] NVIDIA. Nvidia jetsonnano, April 2021.

[80] Kemal Oksuz, Baris Can Cam, Sinan Kalkan, and Emre Akbas. Imbalance problems in object detection: A review. *IEEE transactions on pattern analysis and machine intelligence*, 43(10), 2020.

[81] Omar Besbes, Yonatan Gur, Assaf Zeevi. Stochastic Multi-Armed-Bandit Problem with Non-stationary Rewards . In *NeurIPS*, 2014.

[82] Arthi Padmanabhan, Neil Agarwal, Anand Iyer, Ganesh Ananthanarayanan, Yuanchao Shu, Nikolaos Karianakis, Guoqing Harry Xu, and Ravi Netravali. Gemel: Model merging for memory-efficient, real-time video analytics at the edge, 2022.

[83] Xunyu Pan and Siwei Lyu. Detecting image region duplication using sift features. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1706–1709. IEEE, 2010.

[84] Sibendu Paul, Utsav Drolia, Y. Charlie Hu, and Srimat T. Chakradhar. Aqua: Analytical quality assessment for optimizing video analytics systems, 2021.

[85] Sibendu Paul, Kunal Rao, Giuseppe Coviello, Murugan Sankaradas, Oliver Po, Y. Charlie Hu, and Srimat Chakradhar. Enhancing video analytics accuracy via real-time automated camera parameter tuning. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems*, SenSys '22, page 291–304, New York, NY, USA, 2023. Association for Computing Machinery.

[86] PTZ Optics. PTZ Optics PTZ cameras. https://ptzoptics.com/products/.

[87] PTZ Optics. What is ePTZ and how does it compare with true PTZ? https://ptzoptics.com/what-is-eptz/.

[88] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[89] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, Oliver Wang. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *CVPR*, 2018.

[90] Mohammed Rijas. Powering the edge with ai in an iot world. https://www.forbes.com/sites/forbestechcouncil/2020/04/06/powering-the-edge-with-ai-in-an-iot-world/.

[91] Alberto Rizzoli. 7 Game-Changing AI Applications in the Sports Industry. https://www.v7labs.com/blog/ai-in-sports, 2022.

[92] Francisco Romero, Qian Li, Neeraja J. Yadwadkar, and Christos Kozyrakis. INFaaS: Automated model-less inference serving. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 397–411. USENIX Association, July 2021.

[93] SCW. PTZ Auto-Tracking Explained. https://www.getscw.com/knowledge-base/auto-tracking-explained.

[94] SecurityBros. Which Cheap Outdoor WiFi PTZ IP Camera is Best? Boavision vs Inqmega. https://securitybros.com/which-cheap-outdoor-wifi-ptz-ip-camera-is-best-boavision-vs-inqmega/.

[95] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *NeurIPS*, 2015.

[96] Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, and Ravi Sundaram. Nexus: A gpu cluster engine for accelerating dnn-based video analysis. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, pages 322–337, New York, NY, USA, 2019. Association for Computing Machinery.

[97] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.

[98] Shubham Jain, Viet Nguyen, Marco Gruteser, Paramvir Bahl. Panoptes: Servicing Multiple Applications Simultaneously using Steerable Cameras. In *IPSN*, 2017.

[99] Vishwanath A Sindagi and Vishal M Patel. Generating high-quality crowd density maps using contextual pyramid cnns. In *Proceedings of the IEEE international conference on computer vision*, pages 1861–1870, 2017.

[100] Ryan Steed and Aylin Caliskan. Image representations learned with unsupervised pre-training contain human-like biases. In *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*, pages 701–713, 2021.

[101] Abhijit Suprem, Joy Arulraj, Calton Pu, and Joao Ferreira. Odin: Automated drift detection and recovery in video analytics. *Proc. VLDB Endow.*, 13(12):2453–2465, July 2020.

[102] SV3C. SV3C Security Camera Outdoor . https://www.sv3c.com/.

[103] Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection. *CoRR*, abs/1911.09070, 2019.

[104] Amos Tversky and Itamar Gati. Similarity, separability, and the triangle inequality. *Psychological review*, 89(2):123, 1982.

[105] Deepak Vasisht, Zerina Kapetanovic, Jong-ho Won, Xinxin Jin, Ranveer Chandra, Ashish Kapoor, Sudipta N. Sinha, Madhusudhan Sudarshan, and Sean Stratman. Farmbeats: An iot platform for data-driven agriculture. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI'17, page 515–528, USA, 2017. USENIX Association.

[106] Lluís Martínez; Margarita Cabrera-Bean; Josep Vidal. A Multi-Armed Bandit Model for Non-Stationary Wireless Network Selection. In *2021 IEEE Globecom Workshops*. Ieee, 2021.

[107] Vidit Saxena, Joakim Jaldén, Joseph E. Gonzalez, Mats Bengtsson, Hugo Tullberg, Ion Stoica. Contextual Multi-Armed Bandits for Link Adaptation in Cellular Networks. In *NetAI*, 2019.

[108] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-yolov4: Scaling cross stage partial network, 2021.

[109] Junjue Wang, Ziqiang Feng, Shilpa George, Roger Iyengar, Padmanabhan Pillai, and Mahadev Satyanarayanan. Towards scalable edge-native applications. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, SEC '19, page 152–165, New York, NY, USA, 2019. Association for Computing Machinery.

[110] Yiding Wang, Weiyan Wang, Junxue Zhang, Junchen Jiang, and Kai Chen. Bridging the edge-cloud barrier for real-time advanced vision analytics. In *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, Renton, WA, July 2019. USENIX Association.

[111] Zeyu Wang, Klint Qinami, Ioannis Christos Karakozis, Kyle Genova, Prem Nair, Kenji Hata, and Olga Russakovsky. Towards fairness in visual recognition: Effective strategies for bias mitigation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8919–8928, 2020.

[112] Mengwei Xu, Mengze Zhu, Yunxin Liu, Felix Xiaozhu Lin, and Xuanzhe Liu. Deepcache: Principled cache for mobile deep vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, MobiCom '18, page 129–144, New York, NY, USA, 2018. Association for Computing Machinery.

[113] Yifan Yang, Guorong Li, Dawei Du, Qingming Huang, and Nicu Sebe. Embedding perspective analysis into multi-column convolutional neural network for crowd counting. *IEEE Transactions on Image Processing*, 30:1395–1407, 2020.

[114] Yifu Zhang, Peize Sun, Yi Jiang, Dongdong Yu, Zehuan Yuan, Ping Luo, Wenyu Liu, Xinggang Wang. Bytetrack: Multi-object tracking by associating every detection box, 2021.

[115] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. *SIGCOMM Comput. Commun. Rev.*, 45(4):325–338, aug 2015.

[116] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[117] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, pages 818–833. Springer, 2014.

[118] Anran Zhang, Lei Yue, Jiayi Shen, Fan Zhu, Xiantong Zhen, Xianbin Cao, and Ling Shao. Attentional neural fields for crowd counting. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5714–5723, 2019.

[119] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J. Freedman. Live video analytics at scale with approximation and delay-tolerance. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI'17, pages 377–392, Berkeley, CA, USA, 2017. USENIX Association.

[120] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J. Freedman. Live video analytics at scale with approximation and delay-tolerance. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 377–392, Boston, MA, March 2017. USENIX Association.

[121] Tan Zhang, Aakanksha Chowdhery, Paramvir Bahl, Kyle Jamieson, and Suman Banerjee. The design and implementation of a wireless video surveillance system. pages 426–438, 09 2015.

[122] Kun Zhao, Bin Liu, Luchuan Song, Weihai Li, and Nenghai Yu. Cascaded residual density network for crowd counting. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 2199–2203. IEEE, 2019.

[123] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.

[124] Yuhao Zhu, Anand Samajdar, Matthew Mattina, and Paul Whatmough. Euphrates: Algorithm-soc co-design for low-power mobile continuous vision. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ISCA '18, page 547–560. IEEE Press, 2018.

A Appendix

A.1 Additional objects and tasks

To evaluate MadEye's generality, we ran experiments that target different object types (counting lions and elephants in safari videos using FasterRCNN and SSD) or tasks (find sitting people using the OpenPose [24] pose estimation model). Importantly, MadEye did not require any special tuning, and instead learned the (new) objects and tasks of interest via training on the original models' results. In line with our main evaluation results, MadEye improved accuracies over *best fixed* by 4.6 - 14.5% and 2.8 - 10.9% for lions and elephants respectively and by 9.5 - 17.1% for pose estimation when using 400° per second rotation speeds and a {24 Mbps, 20 ms} network. The differences in improvements can be explained by the dynamics of the videos – elephants in the videos were largely static so *best fixed* performed well; in contrast, the videos used for identifying lions and sitting people had many moving objects which necessitated frequent orientation switches.

A.2 Workloads

Model	Object	Type
SSD	people	aggregate count
Faster RCNN	cars	binary classification
SSD	people	count
YOLOv4	people	detection
Faster RCNN	people	detection

Table 3: Workload 1 (W1)

Model	Object	Type
YOLOv4	people	aggregate count
Tiny YOLOv4	people	aggregate count
Tiny YOLOv4	people	detection
YOLOv4	people	binary classification
Tiny YOLOv4	people	aggregate count
Faster RCNN	people	count
Faster RCNN	people	detection
Faster RCNN	car	count
YOLOv4	people	aggregate count
YOLOv4	people	detection
YOLOv4	people	count
Tiny YOLOv4	people	aggregate count
YOLOv4	car	count
YOLOv4	car	detection
Tiny YOLOv4	car	count
SSD	person	binary classification
Faster RCNN	car	count
SSD	car	count

Table 4: Workload 2 (W2)

Model	Object	Type
SSD	car	binary classification
Faster RCNN	people	aggregate count
Faster RCNN	people	count
Tiny YOLOv4	people	binary classification
Tiny YOLOv4	people	binary classification
Tiny YOLOv4	people	aggregate count
YOLOv4	people	count
Faster RCNN	people	aggregate count
SSD	people	binary classification
Faster RCNN	car	count
SSD	car	count

Table 5: Workload 3 (W3)

Model	Object	Type
Tiny YOLOv4	car	count
Faster RCNN	car	detection
Faster RCNN	people	aggregate count

Table 6: Workload 4 (W4)

Model	Object	Type
Tiny YOLOv4	car	count
SSD	car	count
Faster RCNN	people	aggregate count

Table 7: Workload 5 (W5)

Model	Object	Type
Tiny YOLOv4	people	aggregate count
Tiny YOLOv4	people	binary classification
SSD	car	count
YOLOv4	people	aggregate count
Tiny YOLOv4	people	count
Faster RCNN	car	binary classification
SSD	people	detection
Faster RCNN	car	detection
Faster RCNN	people	aggregate count
YOLOv4	car	count
Tiny YOLOv4	people	aggregate count
Faster RCNN	people	detection
SSD	people	aggregate count
YOLOv4	car	detection

Table 8: Workload 6 (W6)

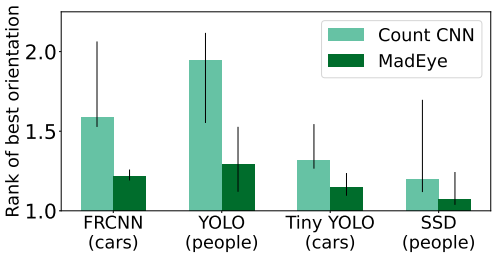


Figure 16: Comparing different approximation model designs: MadEye's lightweight detection models and compressed counting models (Count CNN). Results use all videos, {24 Mbps; 20 ms}, 15 fps, and list median rank assigned to the best explored orientation at each timestep (error bars for 25-75th percentiles).

Model	Object	Type
YOLOv4	people	binary classification
SSD	people	detection
Tiny YOLOv4	car	binary classification
Tiny YOLOv4	people	detection
SSD	people	binary classification
SSD	people	aggregate count
Tiny YOLOv4	people	detection
SSD	car	count
SSD	people	count
Faster RCNN	people	count
YOLOv4	people	count
Faster RCNN	people	binary classification
Tiny YOLOv4	people	aggregate count
Faster RCNN	people	aggregate count
Faster RCNN	car	count
YOLOv4	car	binary classification

Table 9: Workload 7 (W7)

Model	Object	Type
Faster RCNN	car	count
Tiny YOLOv4	people	binary classification
YOLOv4	people	aggregate count
YOLOv4	car	count
Tiny YOLOv4	people	aggregate count
Faster RCNN	people	aggregate count
YOLOv4	people	aggregate count
Faster RCNN	car	count
SSD	car	count
Faster RCNN	car	count
SSD	car	binary classification
YOLOv4	car	binary classification
SSD	car	binary classification
SSD	people	count
YOLOv4	people	count
YOLOv4	car	binary classification
Faster RCNN	person	aggregate count
SSD	car	detection

Table 10: Workload 8 (W8)

Model	Object	Type
Tiny YOLOv4	people	aggregate count
Faster RCNN	people	count
Faster RCNN	people	count
Tiny YOLOv4	car	detection
Tiny YOLOv4	people	binary clasification
YOLOv4	people	detection
Faster RCNN	people	count
YOLOv4	people	aggregate count
SSD	people	aggregate count

Table 11: Workload 9 (W9)

Model	Object	Type
Faster RCNN	people	aggregate count
Faster RCNN	car	count
Faster RCNN	people	count

Table 12: Workload 10 (W10)