Real-time Detection of Activities in Untrimmed Videos

Joshua Gleason
gleason@umiacs.umd.edu

Carlos D. Castillo
carlos.d.castillo@umiacs.umd.edu

Rama Chellappa
rama@umiacs.umd.edu

University of Maryland, College Park

Abstract

Real-time detection of spatio-temporal sparse activities in untrimmed videos is a challenging problem. In this work, we present the details of our proposed solution. We begin with a slow baseline implementation of a previously state-of-the-art system [13] and redesign it to achieve real-time performance for detecting 37 activities in the ActEV19 Sequestered Data Leaderboard [4]. This is primarily achieved by introducing speed related hyperparameters into the baseline approach. A tradeoff analysis is performed to assist in hyperparameter selection which results in a real-time, high quality action detection system. Our system achieves an AUDC score of 0.476 *on the ActEV19 Sequestered Data Leaderboard.*

1. Introduction

The goal of this work is to produce a system for detecting activities in untrimmed videos which runs in real-time with high accuracy on a large number of human and vehicle activities. By real-time we mean that the time between when video files are provided to the system until the time that all detected activities are reported is less than the total run-time duration of the input videos. Or to put it another way, a system with relative time as reported on the ActEV19 Sequestered Data Leaderboard (SDL) of at most one.

In principle, the problem seems similar to action detection on common untrimmed video datasets such as AVA [14], THUMOS [18], and ActivityNet [8]. In practice, systems that perform well on these datasets may not perform as well on untrimmed security videos. For example, a baseline re-implementation of R-C3D [27] submitted to the ActEV19 SDL [4] performed poorly. Qualitatively this behavior may be explained due to particular differences in input domain. For example, videos in MEVA [1] and ActEV19 TRECVID [2] have a very small spatial extent, often spanning less than 5% of pixels. Comparatively, the previously mentioned datasets often contain videos sampled from entertainment sources like movies or internet videos which usually have activities appearing front-and-center with high spatial extent. Similar to the way special considerations are needed for small object detection, special considerations also need to be taken when considering spatially sparse activity detection.

Such domain differences are one motivation for choosing to start with a baseline implementation of the system proposed by Gleason et al. [13]. This system, which shall henceforth be referred to as *the baseline system*, was designed for the DIVA 1.A. dataset, which is a subset of the ActEV19 TRECVID dataset (Section 3). This is a previously state-of-the-art action detection system for untrimmed security videos. Another reason we start with this method is that it utilizes spatial information of activities, but does not require expensive track or object level annotations, instead operating on cuboids. Low spatial precision of cuboid annotations also seems to play a role in making the baseline system robust to annotation errors, which in turn decreases the cost of collecting annotations.

According to the authors of [13], two reasons why the baseline system performs well are: 1) high recall and high quantity of proposals based on object detections, and 2) predictions are based only on optical flow which acts as an implicit background subtraction module, thus significantly reducing the support of the input domain. Unfortunately, these properties are precisely the reason why the system is slow. Even with multiple code optimizations our implementation of the baseline method was more than six times slower than real-time. In order to retain or improve on the high metric performance of the baseline system we opt to keep all the fundamental components, and instead introduce hyperparameters to improve system speed. We perform a tradeoff analysis between speed and metric performance and use this analysis to choose reasonable values for the hyperparameters. This results in a system which is able to operate at real-time, while retaining the majority of the baseline system's performance.

The primary goal of this work is to implement a real-time system for the ActEV19 SDL. The secondary goals are to provide a detailed description of our system and to document this as an example study on taking slow DCNN-

based systems and augmenting them to achieve real-time performance. Our systematic approach can be summarized as:

1. Profile the system, identifying the slowest components.
2. Introduce hyperparameters which can improve performance. For example, downsampling the input, or reducing the number of inputs to a component.
3. Perform a tradeoff analysis of the hyperparameters with respect to speed and performance metric of choice.
4. Choose hyperparameters based on this analysis which achieve the desired speed with minimal degradation to metric performance.

The remainder of the paper is organized as follows: Section 2 describes related works. Section 3 introduces the three datasets used for training and evaluation. Section 4 covers the method and implementation details of our system. Section 5 defines the training loss and describes the steps for training our system. Section 6 describes the experiments for hyperparameter selection, the performance of our system on validation, and ActEV SDL performance. Section 7 consists of some concluding remarks and directions for future work.

2. Related Works

The method proposed in this paper follows a proposal-based approach for action detection. It is reasonable to draw comparisons to the object detection research which has found great success in many practical applications in the last few years [26]. For example, the method we base our work on [13] is similar to the R-CNN object detector [11]. The obvious difference is that instead of acting on a 2D spatial domain, action detection operates in a 3D spatio-temporal domain.

It may at first seem like object detection techniques could be trivially extended to act on an additional dimension, however this is not straightforward. The additional temporal dimension significantly increases the complexity of the input domain. In the case of DCNNs with 3D convolutional layers this has been shown to increase the amount of training samples [15]. Another point of contention is that bounds on the temporal dimension either do not exist (streaming video) or are inconsistent (recorded video). Combining these differences with the lack of high quality spatio-temporal action datasets are some reasons why extending object detection methods to spatio-temporal action detection requires special consideration.

Despite these difficulties, extending object detection methods to action detection is still a common approach to the problem [9, 10, 13, 17, 23, 25]. Most of these methods either do not produce activity instances, instead producing frame level classification [10, 9], or use an alternative method to aggregate multiple activity segments into instances [17, 23, 25]. An alternative approach which has been proposed is to first perform frame-level classification then use temporal linking to produce instances [3]. The attractive feature of the baseline approach [13] is that it directly produces detections of activity instances without additional temporal linking. This is accomplished by uniformly sampling frames across arbitrarily sized proposals. The proposals are produced via hierarchical clustering of object detections.

3. Datasets

In this section we overview the three datasets used for training, hyperparameter tuning, and evaluation. These are the ActEV Sequestered Data, the Multiview Extended Video with Activities (MEVA) [1], and the ActEV19 TRECVID data [2].

3.1. ActEV Sequestered Data

The final evaluation of our system is reported as described by the ActEV Sequestered Data Leaderboard (SDL) [4]. The challenge consists of reporting temporal detections for 37 activities in untrimmed security videos. The videos may be positioned either indoor or outdoor, and a subset of the videos are infrared (IR) video. There are separate leaderboards for electro-optical (EO) videos, i.e. standard digital visible-light videos, and IR videos. The sequestered data used for scoring are not made public, however, an *unlabeled* dataset named MEVA is provided.

The submitted system is required to output each activity instance with an associated start/end frame and detection confidence. Scoring is performed using the metrics described in Section 6.1.

3.2. MEVA

The MEVA dataset contains 2,225 approximately 5 minute videos from 25 EO cameras and 4 IR cameras. Each of the IR cameras is paired with an EO camera positioned in approximately the same location and orientation. The EO videos have resolutions of either 1920x1080 or 1920x1072 and IR videos are 352x240. All videos are timestamped and synchronized via GPS time and all but one camera has fixed orientation. All cameras have a fixed frame-rate of 30 Hz. The single non-fixed orientation camera is in patrol mode, i.e. the camera points at one location for a fixed amount of time, then quickly orients to a new location. Of the 29 cameras, 18 are located outdoors and 11 are indoors, all of the EO-IR camera pairs are outdoors.

The MEVA dataset was collected as a realistic representation of what would be seen by a multi-camera system

Activity	Count	Avg. area %	Avg. duration (frames)
person_enters_through_structure	3995	5.7%	129
person_exits_through_structure	3543	5.4%	134
person_opens_facility_door	2054	4.9%	134
vehicle_turning_left	1766	4.0%	148
Talking	1694	7.1%	682
vehicle_turning_right	1471	2.8%	167
vehicle_stopping	992	4.2%	160
person_picks_up_object	958	3.1%	55
vehicle_starting	906	4.0%	152
person_sitting_down	883	2.2%	65
person_opens_vehicle_door	738	2.2%	70
person_sets_down_object	724	2.3%	50
person_standing_up	667	2.3%	58
person_closes_vehicle_door	610	2.5%	64
person_exits_vehicle	470	2.9%	107
specialized_texting_phone	420	6.4%	767
person_enters_vehicle	411	3.3%	94
vehicle_u_turn	401	4.6%	330
person_closes_facility_door	364	3.6%	85
hand_interaction	349	2.5%	59
vehicle_reversing	339	5.0%	277
object_transfer	321	7.9%	47
Transport_HeavyCarry	321	8.1%	259
specialized_talking_phone	264	7.4%	651
vehicle_drops_off_person	238	5.1%	374
person_person_embrace	211	2.4%	81
Riding	170	7.5%	266
person_purchasing	142	10.1%	356
vehicle_picks_up_person	107	3.7%	329
person_reading_document	84	3.9%	1195
Unloading	77	2.0%	177
Open_Trunk	75	2.4%	106
Closing_Trunk	67	2.6%	95
person_laptop_interaction	35	2.0%	1011
person_loads_vehicle	31	3.5%	168
abandon_package	5	4.6%	169
theft	1	6.5%	242
Per-class Average	**700.1**	**4.3%**	**252**

Table 1. Per-class counts, relative area, and number of frames of annotated activity instances in MEVA dataset. There are a total of $25,904$ annotated instances, of which $22,992$ are selected for training and $2,912$ are selected for validation.

in day-to-day operation. It is not uncommon to find entire videos with nothing occurring, and some activities are exceptionally rare, most notably "Abandon Package" and "Theft". There is also overlap among some camera views, although we don not consider the multiview action detection problem in this work.

Since MEVA is provided without annotations, a collaborative effort was undertaken among DIVA participants to produce annotations for training and internal validation. This effort, while ongoing, has produced annotations for a large portion of MEVA. Details of the MEVA annotations used for training the system described in this manuscript can be found in Table 1. Inspecting these annotations indicates that they are imperfect. This was noted from the observation that after watching approximately 20 minutes of video one is likely to notice at least one false positive and one false negative.

3.2.1 MEVA Annotation Format

Each MEVA annotation contains, at minimum, a spatial axis-aligned bounding box (AABB), start/end frame, and activity label. We will henceforth refer to annotations in this format as *cuboid annotations*. The AABB of each cuboid annotation corresponds to the entire spatial span of an activity within a video, therefore it is possible that an annotation encompasses the entire video frame. For example, if a person rides a bicycle from the lower left corner of a frame to the upper right corner, then the AABB for that "Riding" activity corresponds to the entire frame. The start and end frames are the frame when the bicycle or rider are first visible to first frame where both are no longer visible. For the majority of activities the spatial bounds are more limited than this, with the average activity instance covering only 4.8% of the frame area. Of the videos which are fully annotated, at least one of the 37 activity classes is occurring in only 24.1% of the frames.

3.3. ActEV19 TRECVID

The ActEV19 TRECVID dataset consists of eighteen activity types, of which, eleven are target activities for the ActEV SDL challenge. The dataset is provided in three splits: train, validate, and eval, with annotations being provided only for the train and validation splits. The train dataset contains 64 videos totaling 2 hours 28 minutes with 1,338 instances of the eighteen activity types, of which 609 correspond to the eleven leaderboard activities. The validate dataset consists of 54 videos totaling 1 hour 52 minutes with 1,128 instances of the eighteen activity types, of which 468 correspond to the eleven leaderboard activities.

While this dataset is relatively small compared to MEVA, we make use of it in two ways. First, since it is small and contains high quality labels we utilize it to motivate our hyperparameter tradeoff experiments in Section 6. We also use the eleven activities which correspond to SDL labels by merging the train and validate annotations with the MEVA dataset annotations in order to train the system submitted to the ActEV SDL for final evaluation.

3.3.1 ActEV19 TRECVID Annotation Format

The ActEV19 TRECVID dataset annotations provide object level annotations for each activity instance, where each annotation consists of a track of each object involved in an activity, as well as a start and end time. These tracks are converted to a cuboid format by using the union box of all the track geometry.

4. Method

In this section we describe the details of the system as it would be used during evaluation. The description of the training step is deferred to Section 5. An overview of the steps are:

1. Proposal Generation

(a) Human/vehicle object detection performed on each video.

(b) Hierarchical clustering applied to object detections in the spatio-temporal domain.

(c) Initial action proposal cuboids constructed by taking the minimum spatio-temporal axis-aligned bounding box containing each cluster's detection boxes.

(d) Action proposal cuboids produced by applying "temporal jittering" to initial action proposal cuboids.

2. Proposal Inference

(a) TV-L1 optical flow computed for each video.

(b) Optical flow sampled corresponding to each action proposal cuboid.

(c) Optical flow samples sent through a deep convolutional network to predict an action label (including "No Activity") and temporal refinement regression value.

(d) Predicted class labels assigned to each proposal. Low confidence proposals discarded and temporal refinement prediction applied to remaining action proposals.

(e) 3D non-maximum suppression applied to remaining proposals.

These are the same steps proposed in the design of the baseline system [13]. Owing to space constraints, we only discuss the differences between our system and the *baseline system*, focusing on implementation details that have significant impact on system speed. The reader may notice that some sections have parameters in their titles. These are the hyperparameters introduced to tune system speed. The values for these parameters are explored in Section 6.

4.1. Proposal Generation

The first step is proposal generation which outputs what we call *action proposal cuboids*. The cuboids are analogous to proposals used in object detection algorithms like R-CNN.

4.1.1 Object Detection (T_{det})

The first step in generating action proposal cuboids is object detection. This is performed using a Mask-RCNN detector [16] with a Feature Pyramid Network backbone [20] trained on MSCOCO [21]. To improve speed we only perform object detection once every T_{det} frames. Since all target activities involve humans or vehicles we only keep detections corresponding to "person", "car", "airplane", "bus", "train", "truck", and "boat".

Implementation We use a publicly available PyTorch [22] port of Detectron [12, 24] with the e2e_mask_rcnn_R-101FPN_2x.yaml configuration. Object detection is performed in a separate process for each video where up to two videos may be processed at a time on each available GPU.

4.1.2 Hierarchical Clustering

The next step is to perform hierarchical clustering on the object detections for each video as described in the baseline method [13]. Each object detection is represented as a point in 3D space (x, y, f) which corresponds the center of the detection box in pixels and the frame number. To account for the T_{det} parameter, an additional noise term is added to f so the input points to clustering are $(x, y, f + X)$ where X is a random integer between $-\lfloor T_{det}/2 \rfloor$ and $\lfloor T_{det}/2 \rfloor$ inclusive. Alternatively, we could have scaled f by $1/T_{det}$, however in preliminary experiments we found our method produced slightly higher recall. If there are more than 44,000 detections in a five-minute video we randomly select 44,000 to keep clustering from taking too long. We found that this had effectively no impact on overall action proposal recall.

We have tried other, more sophisticated representations for clustering such as using deep features or additional detection attributes like object aspect ratio, area, and class label. We've also experimented with additional scaling and feature selection of the representations. None of these preliminary experiments produced proposals with better recall than the simple (x, y, f) representation.

Implementation Hierarchical clustering was implemented from scratch in Python 3 using numpy.

4.1.3 Temporal Jittering

The final action proposal cuboids are constructed by applying the temporal jittering algorithm described in the baseline method [13]. No changes were made to this algorithm.

Implementation Temporal jittering was implemented from scratch in Python 3 using numpy.

4.2. Proposal Inference

In this section we describe the steps involved in producing activity predictions from action proposal cuboids.

4.2.1 Optical Flow (T_{flow}, A_{flow})

Optical flow is computed using the TV-L1 algorithm [28]. We introduce two parameters which affect computation speed. First, we sub-sample the RGB frames with a sampling period of T_{flow}. For example $T_{flow} = 2$ indicates

sampling every other frame. Second, we down-sample each RGB frame's area by a factor of A_{flow} while maintaining the aspect ratio. To account for small camera motion and vibrations we also subtract the average flow from each sampled cuboid.

Implementation We use the GPU accelerated TV-L1 optical flow algorithm from OpenCV 2.4 [6] in C++[1]. Optical flow is computed in a separate process for each video where up to two videos may be processed at a time on each available GPU. We found that increasing the number of allowed processes per GPU, while possible, resulted in a decrease in overall speed. Each component of optical flow is scaled and translated to span $[-0.5, 255.5]$, then rounded and cast to uint8. Each component is then saved as a single channel JPEG image. The translation and scaling parameters are stored so that the original flow can be converted back to the proper range upon loading. Experiments with lossless compression showed that the artifacts introduced by JPEG compression and quantization had negligible impact on the system's performance while significantly decreasing memory usage. Since the JPEG images are relatively small we are able to store the compressed optical flow images directly into RAM via temporary file system (tmpfs) which significantly improves save and load times.

4.2.2 Data Sampling (N_{fr}, S_{fr}, F_{fr})

Before proposals can be classified, we need to sample the optical flow corresponding to the cuboids bounds. On average there are about $25,000$ cuboid proposals per five-minute clip in MEVA. For each action proposal cuboid we first extend the smallest spatial dimension to give a aspect ratio of 1, then pad the cuboid by five pixels on all sides to account for object detection errors. We sample this square region from N_{fr} optical flow frames uniformly sampled across the span of the cuboid proposal. The region is then resized to $S_{fr} \times S_{fr}$ pixels using bilinear interpolation. Since optical flow is in units of pixels/frame we take extra care when resizing and apply the same scaling factor to the optical flow values.

When a cuboid spans less than N_{fr} frames then the same frame may be sampled multiple times. When a crop is performed that goes outside the bounds of the flow image we pad with zeros. As an additional speedup we only sample frames which are divisible by F_{fr}. The motivation for this hyperparameters is due to specific implementation details.

Implementation One of the most important parts of the system is the ability to sample optical flow quickly. A naïve approach would be to iterate through each action proposal cuboid, sampling the N_{fr} corresponding frames, then applying cropping and resizing. Unfortunately, while simple to implement, this approach is extremely slow since it must decompress N_{fr} optical flow frames for each action proposal cuboid. Due to the large number of proposals this becomes a significant bottleneck of our system.

To solve this issue, we first collect all the crops for each optical flow frame that will ever need to be taken. Next, we iterate over each frame which has at least one crop associated, load the frame, then compute all the resized crops for that image. The resulting crops are saved back to the temporary file storage and indexed so they can be referred to later by their original spatial dimensions. Since the crops are much smaller than the original flow images this strategy results in an order-of-magnitude speed-up during evaluation. this stage further since it results in the total number of frames visited being reduced by a factor of F_{fr}.

Loading, cropping, and resizing was implemented in Python 3 using the Pillow library. The operations are performed using 32 parallel worker processes where each job produces crops for a single optical flow image.

4.2.3 Network inference

The optical flow associated with each action proposal is center-cropped spatially to $s_{fr} \times s_{fr}$ where

$$s_{fr} := \lceil S_{fr} \cdot 224/256 \rfloor \tag{1}$$

and then sent through a TRI3D network as a $(2, N_{fr}, s_{fr}, s_{fr})$ tensor. For C target classes, the output of the network is a class prediction vector $\hat{y} \in \mathbb{R}^{C+1}$ and two temporal refinement predictions $\hat{v} \in \mathbb{R}^2$ where $\hat{v}_0$ and $\hat{v}_1$ are respectively the predicted start and end temporal refinement values. Following the baseline approach we produce only a single start and end temporal refinement prediction. The final class prediction is taken to be[2]

$$\hat{\omega} := \left(\text{argmax}_{i \geq 1} \hat{y}_i\right) \tag{2}$$

and the predicted final start and end frame are taken to be

$$\left(\hat{f}_{st},\ \hat{f}_{end}\right) := t \cdot \hat{v} + f_a \tag{3}$$

where f_a is the center frame of the proposal and t is half the temporal span of the proposal. The classification confidence score is taken to be

$$p := \text{softmax}(\hat{y})_{\hat{\omega}} := \frac{\exp(\hat{y}_{\hat{\omega}})}{\sum_{i=0}^{C+1} \exp(\hat{y}_i)}. \tag{4}$$

Any proposals with $p \leq \tau_p$ are rejected. For MEVA and the ActEV SDL we choose $\tau_p = 0.07$. For ActEV19 TRECVID we choose $\tau_p = 0.15$ due to containing fewer activity classes. The purpose of this threshold is to reject

[1] OpticalFlowDual_TVL1_GPU parameters: tau = 0.25, lambda = 0.1, theta = 0.3, nscales = 5, warps = 5, epsilon = 0.005, iterations = 300, useInitialFlow = false

[2] subscript indices are 0-based

"No Activity" classes. Alternatively we could simply reject when $\text{softmax}(\hat{y})_0$ is above some threshold or is maximal. In preliminary tests on validation data we had slightly better performance when using our approach which also rejects contested activities. Since the final scoring metric determines thresholds for various operating points we choose not to reject other low confidence predictions.

Implementation The network is implemented in PyTorch and uses a large mini-batch size. The exact mini-batch size depends on hyperparameters N_{fr} and S_{fr}. We use PyTorch's built in `nn.DataLoader` class with twelve parallel workers which allows subsequent batches to be staged while network inference is occurring.

4.2.4 Non-maximum suppression

A simple 3D non-maximum suppression is used to remove overlapping proposals. This algorithm is the same as the one described in the baseline method [13].

Implementation NMS was implemented from scratch in Python 3 using numpy with vectorization. Our system potentially produces many activity predictions prior to NMS. Because of this, it is important that the algorithm is implemented with some consideration for speed. We found that a completely iterative approach using dictionary lookups at each step resulted in significant system overhead, increasing the overall system time by a factor of about 20%.

5. Training

Training the TRI3D network is similar to the method described in the baseline method [13]. We use the same thresholds for producing training proposals, taking care to select for hard negatives which slightly overlap ground truth annotations.

The loss function is

$$L := L_{cls} + \lambda[\omega \neq 0]L_{loc} \tag{5}$$

where classification loss L_{cls} is standard categorical cross-entropy loss which is

$$L_{cls}(\hat{y}, \omega) := -\log\left(\text{softmax}(\hat{y})_\omega\right) \tag{6}$$

and localization loss uses smooth L1 which is

$$L_{loc}(\hat{v}, v) := \frac{1}{2}\sum_{i=0}^{1} \text{smooth}_{L1}\left(\hat{v}_i - v_i\right) \tag{7}$$

where

$$\text{smooth}_{L1}(x) := \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \tag{8}$$

and $[\omega \neq 0]$ is an indicator function which disables L_{loc} for negative training examples.

In the preceding equations $\omega \in \{0, ..., C\}$ is the ground-truth activity class, $v \in \mathbb{R}^2$ is the ground-truth temporal refinement target, and $\lambda > 0$ is a hyperparameter. For all our experiments we use $\lambda = 1$. The temporal refinement target v is computed the same as in the baseline method. We use the same optical flow and data sampling approach described in Sections 4.2.1 and 4.2.2 to represent training proposals.

For network optimization we use I3D weights pretrained on Kinetics-400 [7] except for the two fully-connected layers which are initialized using PyTorch defaults.

The training dataset is balanced so that positive activity classes ($\omega \geq 1$) which appear less frequently than others are repeated so that each class is present in equal numbers. While iterating through the dataset, a "No Activity" sample randomly replaces a positive sample with probability 0.05 for MEVA and and 0.1 for AcvEV19 TRECVID.

Data augmentation is applied in the form of random spatial crops with edge lengths s_{fr} and random horizontal flips for all activities except "vehicle left turn", "vehicle right turn", and "vehicle u-turn".

The network is trained with the Adam optimizer [19] with learning rate $5\text{e}{-4}$, weight decay $5\text{e}{-6}$, and $(\beta_1, \beta_2) = (0.9, 0.999)$. An effective mini-batch size of 400 is used by training in parallel across 10 GPUs. The loss for a mini-batch is computed as the average of Eq. 5 evaluated for each sample in the mini-batch. Training continues until it is clear the network is overfitting. This is usually apparent after less than 40,000 batches. Every 250 batches we evaluate loss and per-class classification accuracy on a subset of our validation set consisting of 5,000 labeled proposals, half of which correspond to valid activities and half hard negatives. The system usually reaches its minimum validation loss between mini-batch 5,000 and 20,000. Multiple attempts to investigate and resolve the overfitting observed during training have not led to any concrete conclusions or solutions.

6. Experiments & Results

In this section we described the experiments performed on both the MEVA and ActEV19 TRECVID datasets. We begin by reporting speed versus performance on the TRECVID validation data and then follow up with results on the MEVA validation and ActEV SDL leaderboard.

6.1. Metrics

A full description of the metrics used may be found in the ActEV19 SDL and TRECVID evaluation plans [4, 5]. For ActEV19 TRECVID validation performance, we report probability of miss with at rate-of-false-alarm of 0.15 (p-miss @ 0.15 rfa). For the ActEV19 SDL and MEVA validation results we report probability of miss with at time-based-false-alarm of 0.04 (p-miss @ 0.04 tfa) as well as

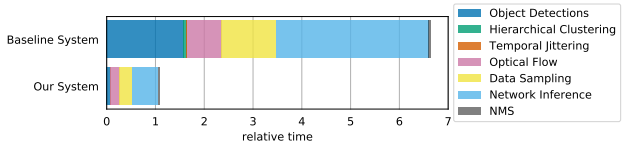

Figure 1. Profile of system speed for the baseline system [13] versus our system with fast hyperparameter choice on the ActEV19 TRECVID validation set. Test hardware used to profile these systems had 4x NVIDIA® GeForce® 1080 Ti GPUs, Intel® Xeon® CPU E5-2683 v4 @ 2.10GHz, and 128GB RAM. This is a similar hardware configuration to the target machine for the ActEV SDL evaluation server. Units in this plot are relative performance with respect to real-time where 1 corresponds to the total run-time of all videos in the dataset. This figure shows that the slowest parts of the system are Object Detection, Optical Flow, Data Sampling, and Network Inference.

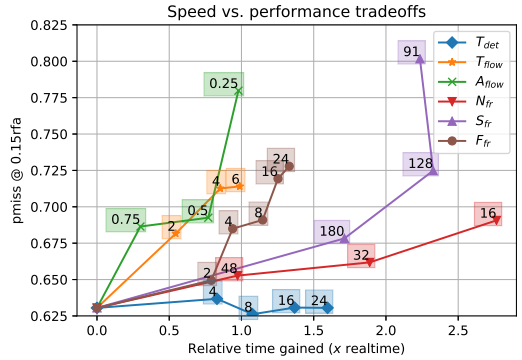

Figure 2. Results of varying hyperparameters on the ActEV19 TRECVID validation set. Each curve represents the results of varying a single hyperparameter with all others kept at the baseline value. The numeric label at each point refers to the value of the varied hyperparameter. Tradeoff is represented as time gained versus p-miss @ 0.15rfa. Time gained is reported in units of "× real-time" which refers to reduction in time as compared to the baseline system in Figure 1. For example, time gained of 1 × real-time corresponds to an overall system performing at 5.6 × real-time.

the normalized area under the DET curve from 0 to 0.2 tfa (AUDC).

6.2. Speedups

The first step in achieving our goal of real-time performance on the ActEV SDL evaluation server is to profile the baseline system. The top bar in the in Figure 1 shows the measured relative time of each stage of the system. The results of profiling are the primary motivation for introducing the hyperparameters described in the sub-section titles of Section 4. These hyperparameters are as follows:

- $T_{det}(1)$: Object detection input frame sampling period.
- $T_{flow}(1)$: Optical flow input frame sampling period.
- $A_{flow}(1)$: Optical flow input area scale.
- $N_{fr}(64)$: Number of frames sampled from each cuboid and provided to the TRI3D DCNN.
- $S_{fr}(256)$: Sampled flow edge scaling factor.
- $F_{fr}(1)$: Frame number factor. When sampling optical flow, round the frame number to nearest multiple of F_{fr}.

The numbers in parenthesis are the baseline values which correspond to parameters used by the baseline system. These are equivalent to the system proposed in the original work [13]. Each of the parameters profoundly impacts on one or more of the slowest components of the system.

In order to determine how best to assign the parameters, we vary one at a time, keeping the others at baseline values. We then measure how each parameter affects overall speed and metric performance. The results of these experiments were used to inform our decision on the hyperparameter values chosen for training on MEVA and submission to the ActEV SDL, which requires real-time performance as a prerequisite for participation.

To ensure the metrics are not representing the results of a domain shift between the training and validation data we retrained the model for each experiment, using the same hyperparameters during training and validation. The results of the experiments are shown in Figure 2 where each point on the plot refers to a different configuration trained and evaluated on the ActEV19 TRECVID validation set.

We would like to point out that the results of this experiment only provide an approximate upper bound on our choice of hyperparameters. It is outside the scope of this work to attempt to predict how well a particular combination will perform without running the experiment. That said, the results make it clear that choosing, for example, $S_{fr} = 91$ or $A_{flow} = 0.25$ would have a catastrophic impact on p-miss performance. On the other hand p-miss is stable for a wide range of T_{det} values.

Another observation is that the hyperparameters are not independent of each other with respect system speed or performance. For example, if $T_{flow} = 4$ and $F_{fr} = 2$ then optical flow only generates frames which are multiples of 4. Therefore rounding to the nearest available multiple of 2 is identical to sampling the nearest frame, so the system speed and performance would be the same as if $F_{fr} = 1$ or 4.

Due to these complex interactions we use the results in Figure 2 to inform our decision, however the final decision is to some degree based on intuition. The final hyperparameter values are: $T_{det} = 30$, $T_{flow} = 4$, $A_{flow} = 0.75$, $N_{fr} = 32$, $S_{fr} = 164$, $F_{fr} = 8$.

With these values the overall system time decreased from 6.65 to 1.08 times real-time, a speedup factor of 6.15, and p-miss @ 0.15 rfa increased from 0.630 to 0.718, an overall increase of 13.8%. Since the evaluation server uses newer

System	AUDC	p-miss@0.04 tfa	Relative time
UMD (Ours)	**0.476**	**0.545**	0.725
UCF-P	0.604	0.642	0.293
Purdue	0.655	0.737	**0.124**
STARK	0.821	0.857	0.757
INF_ MEVA1	0.930	0.945	0.588

Table 2. Results of our system versus others on ActEV19 SDL for EO task as of December 15 workshop submission deadline.

System	AUDC	p-miss@0.04 tfa	Relative time
UCF-P	0.444	0.530	0.403
UMD (Ours)	**0.476**	**0.545**	0.725
Edge-Intelligence	0.628	0.755	0.939
Purdue	0.642	0.733	**0.272**
STARK	0.717	0.777	0.793
INF_ MEVA1	0.859	0.880	0.922

Table 3. Results on ActEV19 SDL for EO task as of the January 10 snapshot deadline.

System	AUDC	p-miss@0.04 tfa	Relative time
UMD+UCF	**0.416**	**0.494**	1.154
UCF-P	0.438	0.524	0.438
UMD (Ours)	0.476	0.545	0.725
Edge-Intelligence	0.628	0.755	0.939
Purdue	0.642	0.733	**0.272**
INF_ MEVA1	0.680	0.780	0.928
STARK	0.717	0.777	0.717

Table 4. Results on ActEV19 SDL for EO task as of Feburary 1. UMD+UCF is the result of late fusion of our system and UCF-P.

generation GPUs we (correctly) predicted the relative speed for this choice of hyperparameters to surpass real-time during independent evaluation.

6.3. MEVA & ActEV19 SDL

For submission to the ActEV19 SDL, we train our system with the hyperparameter values presented in Section 6.2. We construct the training data by merging the MEVA training dataset described in Section 3 with the overlapping activity types from ActEV19 TRECVID training and validation data. The performance on our internal validation set had a p-miss @ 0.04 tfa of 0.563 and AUDC of 0.492. The true test of our systems ability to generalize comes from the results on the ActEV19 SDL which is performed by an independent evaluator on sequestered data. The results of ActEV19 SDL snapshots at different dates are reported in Tables 2, 3, and 4. The final AUDC of our system is 0.476 with a relative time of 0.725, which exceeds the real-time requirement. The per-class AUDC scores are provided in Figure 3. We believe that the performance of our system on sequestered data is compelling evidence of the utility and generalizablility of our system. Our system also has best AUDC on the IR portion of the ActEV SDL, despite not being trained on any IR data. Further discussion of the IR SDL results are omitted due to space constraints.

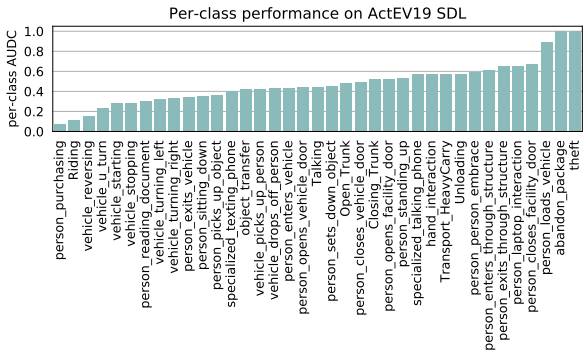

Figure 3. Per-class performance for our submission to the ActEV19 SDL [4] on EO task.

7. Conclusion

Our objective in this work was to make our baseline implementation of Gleason et al. [13] real-time and be effective for a larger set of activities. We evaluated the baseline system and realized that the same features that made the baseline good were the features that made it slow. We defined a set of parameters to accelerate the baseline approach with limited loss of metric score. This was done across datasets and activities: the parameters were evaluated on the ActEV19 TRECVID and the final system was evaluated on the ActEV19 SDL. The speed and metric performance of our system has been confirmed extensively by third party through sequestered evaluation for the ActEV19 SDL.

The careful measurements we have performed in this work may enable future improvements in terms of speed of untrimmed activity detection. Computing proposals that obtain the same recall but have better precision would be very important because the network inference time accounts for half the total processing time. Therefore, obtaining the current level of recall with significantly less proposals would improve the overall system execution time. On the other hand, upon a quick review of the baseline method the object detection and flow computation may appear like onerous steps. We have shown that in a well tuned implementation these account for only a small fraction of the execution time.

Rare and surprise activities such as abandon package and theft for which we do not have many samples will also be part of our future focus.

Acknowledgments

This research is based upon work supported by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via IARPA R&D Contract No. D17PC00345. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

References

[1] The multiview extended video with activities (meva) dataset. https://mevadata.org/, Accessed: 2020-01-25.

[2] Trec video retrieval evaluation (trecvid). https://trecvid.nist.gov/, Accessed: 2020-01-25.

[3] S. Aakur, D. Sawyer, and S. Sarkar. Fine-grained action detection in untrimmed surveillance videos. In *2019 IEEE Winter Applications of Computer Vision Workshops (WACVW)*, pages 38–40. IEEE, 2019.

[4] N. ActEV Team. Draft of actev sequestered data leaderboard (sdl) evaluation plan. https://actev.nist.gov/sdl, Accessed: 2019-12-10.

[5] N. ActEV Team. Draft of trecvid2019 actev evaluation plan. https://actev.nist.gov/trecvid19, Accessed: 2020-01-30.

[6] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[7] J. Carreira and A. Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4724–4733, 2017.

[8] B. G. Fabian Caba Heilbron, Victor Escorcia and J. C. Niebles. Activitynet: A large-scale video benchmark for human activity understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 961–970, 2015.

[9] C. Feichtenhofer, H. Fan, J. Malik, and K. He. Slowfast networks for video recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6202–6211, 2019.

[10] R. Girdhar, J. Carreira, C. Doersch, and A. Zisserman. Video action transformer network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 244–253, 2019.

[11] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 580–587, 2014.

[12] R. Girshick, I. Radosavovic, G. Gkioxari, P. Dollár, and K. He. Detectron. https://github.com/facebookresearch/detectron, 2018.

[13] J. Gleason, R. Ranjan, S. Schwarcz, C. D. Castillo, J.-C. Cheng, and R. Chellappa. A proposal-based solution to spatio-temporal action detection in untrimmed videos. *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 141–150, 2018.

[14] C. Gu, C. Sun, S. Vijayanarasimhan, C. Pantofaru, D. A. Ross, G. Toderici, Y. Li, S. Ricco, R. Sukthankar, C. Schmid, and J. Malik. AVA: A video dataset of spatio-temporally localized atomic visual actions. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[15] K. Hara, H. Kataoka, and Y. Satoh. Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet? In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 6546–6555, 2018.

[16] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2961–2969, 2017.

[17] R. Hou, C. Chen, and M. Shah. Tube convolutional neural network (t-cnn) for action detection in videos. In *The IEEE International Conference on Computer Vision (ICCV)*, 2017.

[18] Y.-G. Jiang, J. Liu, A. Roshan Zamir, G. Toderici, I. Laptev, M. Shah, and R. Sukthankar. THUMOS challenge: Action recognition with a large number of classes. http://crcv.ucf.edu/THUMOS14/, 2014.

[19] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[20] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2117–2125, 2017.

[21] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision (ECCV)*, pages 740–755. Springer, 2014.

[22] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[23] A. J. Rana, P. Tirupattur, M. N. Rizve, K. Duarte, U. Demir, Y. Rawat, and M. Shah. An online system for real-time activity detection in untrimmed surveillance videos. In *The NIST TRECVID Workshop*, 2019.

[24] roytseng tw. Detectron.pytorch. https://github.com/roytseng-tw/Detectron.pytorch, 2018.

[25] G. Singh, S. Saha, M. Sapienza, P. H. Torr, and F. Cuzzolin. Online real-time multiple spatiotemporal action localisation and prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3637–3646, 2017.

[26] X. Wu, D. Sahoo, and S. C. H. Hoi. Recent advances in deep learning for object detection. *ArXiv*, abs/1908.03673, 2019.

[27] H. Xu, A. Das, and K. Saenko. R-c3d: Region convolutional 3d network for temporal activity detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[28] C. Zach, T. Pock, and H. Bischof. A duality based approach for realtime tv-l 1 optical flow. In *Joint Pattern Recognition Symposium*, pages 214–223. Springer, 2007.