Clownfish: Edge and Cloud Symbiosis for Video Stream Analytics

Vinod Nigade, Lin Wang, Henri Bal
VU Amsterdam

***Abstract*—Deep learning (DL) has shown promising results on complex computer vision tasks for video stream analytics recently. However, DL-based analytics typically requires intensive computation, which imposes challenges to the current computing infrastructure. In particular, cloud-only solutions struggle to maintain stable real-time performance due to the streaming over the best-effort Internet, while edge-only solutions require the DL model to be optimized (e.g., pruned or quantized) carefully to fit on resource-constrained devices, affecting the analytics quality.**

In this paper, we propose Clownfish, a framework for efficient video stream analytics that achieves symbiosis of the edge and the cloud. Clownfish deploys a lightweight optimized DL model at the edge for fast response and a complete DL model at the cloud for high accuracy. By exploiting the temporal correlation in video content, Clownfish sends only a subset of video frames intermittently to the cloud and enhances the analytics quality by fusing the results from the cloud model with these from the edge model. Our evaluation based on a system prototype shows that Clownfish always runs in real time and is able to achieve analytics quality comparable to that of cloud-only solutions, even under highly variable network conditions. Clownfish is generally applicable to all video stream analytics tasks that can leverage temporal correlations.

I. Introduction

Video stream analytics has become a fundamental component in various applications in the mobile and Internet-of-Things (IoT) era [1]–[5], [5]–[11]. Applications such as augmented reality and traffic monitoring use cameras to understand the environment. These cameras generate continuous streams of high-quality video data, which has to be analyzed to provide insights by performing computer vision tasks. For example, the traffic monitoring application recognizes car license plates and monitors car trajectories using object recognition and tracking techniques [12]. In many cases, video stream analytics has to be carried out in real time, such that timely reactions can be performed [3], [6], [7], [13], [14]. Recently, deep learning (DL) has become a dominant approach for video stream analytics [15]–[18]. While achieving higher accuracy compared with traditional approaches, DL-based approaches are generally more computation intensive [19]. This further aggravates the challenge of real-time video stream analytics.

Current practices for DL-based video stream analytics are either cloud- [6], [7] or edge-only [3], [13], but none of them are a silver bullet. In cloud-only solutions, the video data has to be continuously streamed to a remote cloud via a wide area network (WAN) before being processed. However, video stream analytics applications typically require a sufficiently high frame rate and low latency [7], [20]. This is challenging in the cloud-only setup, given the high bandwidth variations and jitter that are omnipresent in WAN and wireless and cellular networks [7], [21], [22]. When the network performance drops, the analytics quality will be degraded accordingly.

Alternatively, edge-only solutions propose to deploy computing devices at the network edge and carry out video stream analytics directly from the edge [2]. Since the computation is now performed in close proximity of the video source, the network-related issues can be avoided. However, embedded edge devices (e.g., microcontrollers or NVIDIA Jetson boards), due to their limitations of physical space or energy efficiency, are typically resource-constrained [23], [24]. Thus, DL models have to be optimized or compressed to fit on these devices. The popular model optimization techniques include input resizing, network pruning, data quantization, and model distillation [23]–[27]. However, applying these techniques without affecting the analytics accuracy is challenging, which depends on various factors such as the choice of the optimization method and the model complexity [28] and typically requires tedious parameter tuning in a large parameter space.

In this paper, we propose a hybrid approach to overcome these limitations and present Clownfish – a novel video stream analytics framework that achieves edge-and-cloud symbiosis. Our motivation is to leverage the temporal correlations observed in video content, where the analytics result for a video frame (or a batch of frames) can be used to improve the analytics for later (batches of) video frames if there are strong temporal correlations in the video content. Based on this observation, Clownfish runs an optimized DL model at the edge for real-time responses and enhances the edge analytics results by fusing the generally more accurate analytics results that are intermittently obtained from the cloud running a complete (unoptimized) DL model for the same analytics task. Since the cloud model is only applied on a small subset of the video frames, Clownfish relaxes the network requirement in the cloud-only solution dramatically, with significantly improved analytics quality compared with the edge-only solution. Overall, Clownfish reaps the benefits of both the edge and the cloud through its hybrid design.

Achieving harmonic edge-and-cloud symbiosis is non-trivial and the challenges manifest mainly in the following key questions: (a) which video frames to send to the cloud and (b) how to fuse the cloud analytics results, which are intermittent and possibly delayed to achieve high overall analytics accuracies. To address these challenges, Clownfish provides a systematic methodology which features the following designs.

First, we define a *similarity score* to estimate the temporal correlations between two analytics inputs. The similarity score is calculated with a learning-based approach where we learn directly on the feature vectors produced by the DL model at the edge. Second, we identify application contexts based on the similarity score and apply a context-aware periodical filtering mechanism to make decisions on which video frames to send to the cloud. Third, we use an exponential smoothing function, parameterized with the similarity score, for the fusion of analytics results from the edge and cloud models. Finally, Clownfish includes a confidence-aware design where the exponential smoothing function for result fusion is weighted by the analytics confidence. This is because in some cases, fusing cloud analytics results may even degrade the overall accuracy, as some of the cloud analytics results show low confidence.

In summary, this paper makes the following contributions.

- We propose a novel framework for real-time video stream analytics, which exploits the advantages of both the edge and cloud platforms, simultaneously.
- We provide a systematic methodology to achieve edge-and-cloud symbiosis based on the concept of similarity score, which achieves high overall analytics accuracy by carefully selecting video frames to send to the cloud and fusing the intermittent and possibly delayed analytics results from the cloud.
- We build a system prototype for Clownfish and evaluate its performance through extensive experiments with real-world applications. Our experimental results show that Clownfish is able to maintain real-time performance similar to the edge-only solution while achieving accuracies comparable to the cloud-only solution (accuracy gap within around 2%), under varying network conditions. Clownfish also outperforms existing filtering-based approaches on both accuracy and throughput.

The rest of the paper is organized as follows. Section II presents background and our motivation. Section III presents our system design. Section IV explains our results fusion method. Section V provides our system implementation details and Section VI shows our experimental results. Section VII discusses related work. Section VIII describes challenges and future work, and section IX concludes the paper.

II. Background and Motivation

In this section, we introduce the background information and motivate our work.

A. Video Stream Analytics

With the widespread adoption of cameras in various environments, video stream analytics has become a fundamental component of a variety of mobile and IoT applications, including augmented reality, traffic monitoring, and public safety at airports [29]. These applications typically require high analytics performance with respect to both the execution latency and the analytics accuracy [30]. In essence, video stream analytics applies a computer vision algorithm (e.g., for object detection) on video frames. Traditional video stream analytics pipelines employ algorithms that are based on handcrafted features, where a pre-defined feature vector is first computed on each target frame and then fed into a classifier like a support vector machine (SVM), which finally produces the analytics result [31]. Overall, approaches based on handcrafted features cannot match human performance on accuracy due to factors such as bad feature selection [32].

The real takeoff of applying video stream analytics in IoT applications largely attributes to the remarkable advancement of deep learning in recent years [33]. With deep learning approaches, a deep neural network (DNN) is trained offline with labeled datasets, which is later used at runtime for conducting end-to-end inference [32]. Compared with approaches based on handcrafted features, DNNs produce much higher accuracies in general but require more computations due to the dramatically increased complexity. Nowadays, high-performance accelerators like GPUs and TPUs are employed for DNN-based video stream analytics [19].

While computer vision tasks like object/face recognition only perform per-frame inference, a considerable body of tasks such as identifying human actions, gestures, or activities, take as input a (consecutive) set of frames, i.e., a *frame window*. In this case, the temporal features that capture motion information across multiple video frames are equally important as the spatial features [34]. The window-based inference required in those motion-related tasks deteriorates the computational performance issue due to several reasons: More video frames have to be processed in real-time; preprocessing becomes more complex (e.g., involving expensive optical flow calculation); and the DNN itself becomes heavier with the introduction of the temporal dimension. For example, the state-of-the-art action recognition method I3D (Inception-v1 3D) takes around 274 ms to process a window of 64 frames of size 224×224 on the NVIDIA Titan-X GPU, while this number is only around 10 ms (Inception-v1) for image recognition on a single frame. Overall, the increased computation complexity imposes a critical challenge for real-time video stream analytics.

B. Cloud vs. Edge Computing

Video stream analytics is mostly performed in the cloud due to its high computational demands [19], [36]. One critical challenge in the cloud-based setup concerns the streaming part where a large volume of high-quality video data has to be continuously transmitted to the cloud over the wide area network (WAN). Unfortunately, the growth of the WAN bandwidth has not kept up with the pace of the application bandwidth requirement in recent years [37]. This mismatch makes the network become a bottleneck in many cloud-based solutions. Even worse, WAN bandwidth is not only scarce but also relatively expensive and highly variable [21], [22], [38], [39]. Researchers have explored how to make tradeoffs between bandwidth consumption and analytics accuracy through adapting the quality (including frame resolution and frame rate) of the video stream [36], [39]. Nonetheless, the analytics accuracy suffers inevitably in case of poor network conditions, rendering the overlying applications, especially those mission-

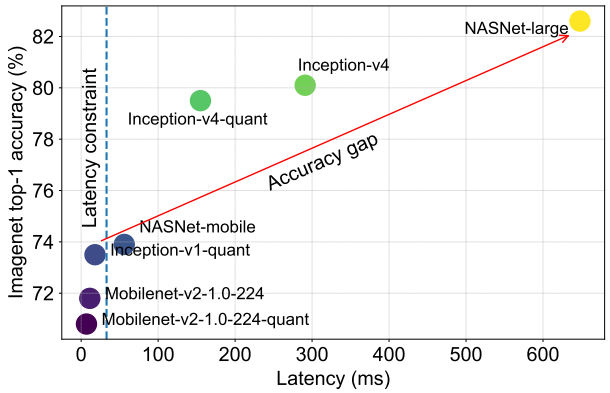

Fig. 1: Model accuracy vs. execution time of complete and optimized (with 8-bit integer quantization) models for image classification with TensorFlow Lite on Pixel 3 (Android 10), reported in [35]. The numbers are based on model executions on the CPU using NNAPI (Neural Network Android API).

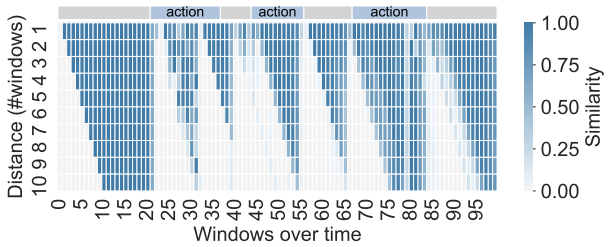

Fig. 2: The similarity of recognition results between windows in a video from the PKU-MMD dataset [42], with the distance between the two windows under comparison varying from 1 to 10 windows.

critical ones like public safety, unreliable. Techniques such as filtering [40] and early discarding [8] have also been explored, but they are only applicable in certain environments where a clear pattern of contextual irrelevance can be observed.

Recently, edge computing has gained traction as an alternative to the cloud-based setup to overcome network-related issues [41]. Edge computing proposes to deploy a small amount of computing resources (e.g., co-locate an NVIDIA Jetson board with a camera) in situ and perform data analytics in close proximity of the data source. Due to physical constraints and economic concerns, edge resources are usually limited compared to the centralized cloud. As a result, edge deployment of computationally expensive DNNs that rely on high-end accelerators for real-time data analytics becomes infeasible. One way of dealing with this issue is to optimize the DNN through techniques including compression, pruning, quantization, or distillation [23]–[27]. While mitigating the resource constraint issue, these techniques are typically lossy especially for complex DNN models, leading to degradation to the analytics quality, as shown in Figure 1.

The above discussion leads us to ask the following question: *How can we reap the benefits of both cloud and edge computing* if neither of them are a silver bullet to real-time video stream analytics? More specifically, instead of treating them as exclusive extremes in the solution space, can we achieve a symbiosis where we combine both the *low latency* property of edge computing and the *high analytics quality* property of cloud computing? Our goal is to answer this question and to propose a framework to simplify the process of achieving an edge-and-cloud symbiosis in applications that are based on video stream analytics.

C. Leveraging Temporal Correlations

One of the salient features of video streams is that the video content has a noticeable temporal correlation across video frames. For example, a human action may span several seconds, which is present in all the video frames captured in the whole action duration. Moreover, the window-based method for handling motion-based tasks, as discussed in Section II-A, further enhances this temporal correlation since neighboring windows can contain a significant number of common frames. To confirm this observation, we perform the action recognition task over a set of videos from the PKU-MMD dataset [42] and calculate the similarity (here we use Cosine similarity) of the recognition results between two windows with varying distances (in the number of windows). The results in Figure 2 clearly show that there are high correlations in windows, even for large distances when the windows fall into the same context.

Our motivation is to leverage such temporal correlations in video streams for achieving an edge-and-cloud symbiosis. The high-level idea is to run an optimized model at the edge with less accurate results and enhance the results with those intermittently obtained from a more accurate model running in the cloud. Thanks to the temporal correlations, we only need to send a selected subset of frames to the cloud while obtaining significant analytics quality improvements. This largely relaxes the bandwidth constraints as in the cloud-based solution.

III. System Design

In this section, we present our design of Clownfish, a novel framework for achieving edge-and-cloud symbiosis for real-time video stream analytics. We denote by "edge" a small computing unit (e.g., an NVIDIA Jetson board) co-located at the video source and by "cloud" a remote data center that is accessible via the WAN. The goal of Clownfish is to achieve inference accuracies comparable to the cloud-based solution even under highly variable conditions on the WAN.

A. Overview

An overview of the Clownfish architecture is depicted in Figure 3. Clownfish consists of system modules spanning across both the edge and the cloud. On the edge side, a module called Window Manager takes the original high-quality video frames from the video source (e.g., a camera) and converts the frames into windows, each containing a fixed number of consecutive frames, as required by the analytics task. The windows are then fed into two components simultaneously:

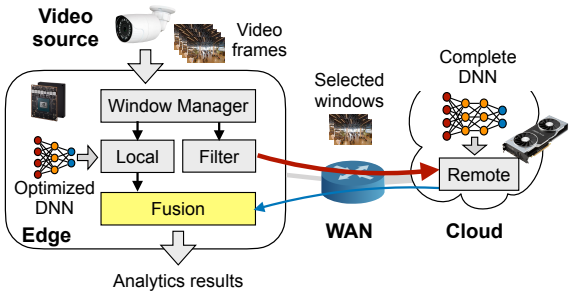

Fig. 3: Overview of the Clownfish architecture.

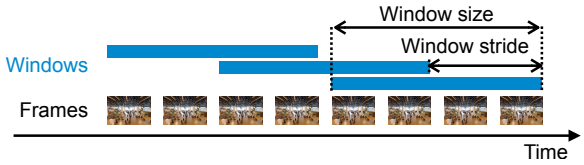

Fig. 4: Windows generated by the Window Manager with the window size and the window stride shown.

Local and Filter. The Local module runs an optimized DNN (called the *local model* from now on) that meets the latency and throughput requirements for the underlying analytics task and outputs the *local results* for each window. The Filter module runs a control policy and sends windows selectively to the cloud. The Remote module deployed on the cloud runs a complete DNN (called the *remote model* from now on), which generally produces more accurate results for the same analytics tasks than the local model, but it is only applied on the small set of filtered windows. The *remote results* (generated by the remote model) will be sent to the Fusion module. The Fusion module employs a fusion method to combine the local and remote results and updates the state of the fusion method upon receiving a new result record from the Remote.

B. Window Manager

The Window Manager module is responsible for two tasks: (a) receiving a stream of video frames captured at a fixed frame rate (e.g., 30 FPS) by the video source and (b) packaging the received frames into windows. We employ a sliding window scheme to generate windows where we maintain a working slot of a fixed size. When new frames arrive, we remove the oldest frames from the tail of the slot and append the newly arrived frames to the head of the slot to generate a new window. There are two parameters, namely the *window size* and the *window stride*, that dictate the efficiency of the sliding window scheme. As illustrated in Figure 4, the window size decides how many frames we should keep in a window for one model run, while the window stride tells how many frames we can jump over before we generate a new window, i.e., the window inter-arrival. Deciding these parameters is non-trivial and we will discuss how to set them up in Section VI.

C. Local and Remote

The Local module (on the edge) takes windows as input from the Window Manager and runs an optimized DNN for the analytics task. The optimized DNN can be generated using the techniques mentioned in Section II-B. Since edge resources are limited, it is essential to ensure that the optimized DNN achieves latencies and throughputs that match the application requirement. Under such a constraint, the DNN with the highest accuracy should be employed. The Remote module (on the cloud) runs a complete DNN for the same analytics task, taking a subset of windows selected by Filter. Unlike Local, Remote does not impose any latency or throughput constraints on the window processing. In other words, the results returned by Remote to the Fusion model can be delayed, intermittent, or both, depending on the selected DNN.

D. Filter

The Filter module takes windows from the Window Manager as input and decides which windows to send to Remote, following a pre-configured window filtering policy. More specifically, we propose a context-aware periodic policy which periodically sends a window to Remote within the same context and restarts the periodic timer at the context transition point. For periodic sending, we set an interval n_w which represents the number of windows to skip before we send another window. This interval should be set carefully since a small n_w will result in high network bandwidth consumption while a large n_w can lead to poor analytics quality, especially in short contexts, due to the lack of remote feedbacks. We will examine the tradeoff on the value selection of n_w in detail in Section VI-C. Since our policy will also send a window when a context starts, the context transition point has to be identified. To this end, we leverage the similarity score where we consider it is a transition point when the change in the similarity score is beyond a threshold, i.e., $\rho_t - \rho_{t-1} \geq 0.5$.

E. Fusion

The Fusion module takes both the local and remote results, and performs a fusion method to integrate these results. More specifically, Fusion leverages the temporal correlations across windows and makes decisions on how much to take from the current local result and how much to propagate from the most recent remote result. As we mentioned already, we define a metric called similarity score to characterize the temporal correlation between windows and design a small neural network (with negligible runtime overhead) to derive this metric at runtime. Based on the similarity score, we apply an exponential smoothing technique to fuse the results from the two sources. We detail the design of the fusion method in the next section.

IV. The Fusion Method

The Fusion module employs a fusion method which aims to improve the local results with the intermittent help from the remote model. Since the Fusion module runs on the edge in parallel to the local model, it is essential that the fusion method is lightweight such that it does not take too many resources from the local model. The main motivation behind our fusion method is that if there is enough temporal correlation in

TABLE I: List of Used Notations

Symbol	Description
w_t	Window at time t
$\vec{p}_l(t)$	Local result for w_t
$\vec{p}_r(t)$	Remote result for w_t
$\vec{p}_f(t)$	Fused result for w_t
h_s	Input video stream rate (frames per second)
h_l	Throughput of the local model (windows per second)
h_r	Throughput of the remote model (windows per second)
ρ	Similarity score produced by SimiNet
$g(\cdot)$	Aggregator that combines local and remote results
α_t	Correlation parameter
β_t	Accumulative remote influence
n_w	Filter interval defined in number of windows
d	Lag of the remote result defined in number of windows

the window sequence, which is true for motion-related video analytics tasks as we have already discussed in Section II-C, the analytics results on past windows can be used to help with the analytics for later windows. Similar techniques have been exploited in time-series forecasting where the current value is estimated using past, often noisy, observations [43].

A. Fusion with Exponential Smoothing

Our fusion method is based on a variant of exponential smoothing, which is powerful yet easy to implement. The analytics result for a window can be represented by a probability vector which contains predicted probability scores on a given list of possible targets (e.g., object types for object recognition or action types for action recognition). We treat the continuous analytics results as a multi-variate time series of probability vectors. Exponential smoothing combines past probability vectors with the current one using a weighted moving average. Denote by $\vec{p}_f(t-1)$ the fused result for the past window w_{t-1} (which is maintained as an internal state of the fusion method) and by $\vec{p}_l(t)$ the local result for window w_t. The fused result $\vec{p}_f(t)$ for w_t can be calculated as

$$\vec{p}_f(t) = \alpha_t \vec{p}_f(t-1) + (1-\alpha_t)\vec{p}_l(t) \tag{1}$$

where $\alpha_t \in [0,1]$ is a correlation parameter that decides the tradeoff between the previous fused result and the current local result. The value for parameter α_t is chosen based on the real-time temporal correlation in the application context.

Now, the question is how to maintain and update the internal state $\vec{p}_f(t-1)$, incorporating the remote results denoted by $\vec{p}_r(t)$. To relax the bandwidth requirement, we send only a subset of the windows to the remote model, leading to intermittent feedbacks from the remote model. The remote results can also be delayed because of a large network latency and/or a long processing time of the remote model (which is more computation-intensive than the local model and may take more time even if running on high-end accelerators [44]). To cope with these conditions, our fusion method introduces two main procedures, namely FUSE and REINFORCE, which take care of the real-time results fusion and the state update upon receiving a remote result, respectively.

The FUSE procedure performs real-time fusion by combining the maintained state $\vec{p}_f(t-1)$ and the available result

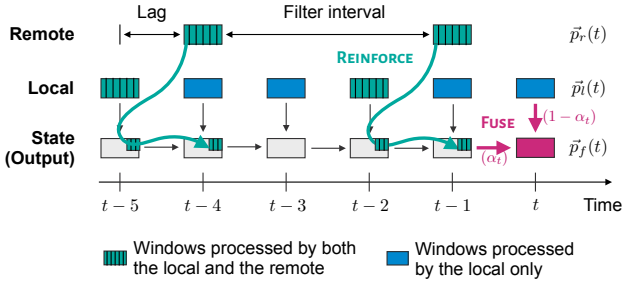

Fig. 5: The two main procedures in our fusion method: FUSE generates output $\vec{p}_f(t)$ by combining the current local result $\vec{p}_l(t)$ with the current state $\vec{p}_f(t-1)$ using a weighted exponential smoothing function. REINFORCE applies retrospective updates on the state $\vec{p}_f$ upon receiving the (intermittent and possibly delayed) remote result $\vec{p}_r$.

$\vec{p}(t)$ for the current window w_t following the exponential smoothing technique as just discussed, i.e.,

$$\vec{p}_f(t) = \begin{cases} \vec{p}(t), & \text{if } t = 1, \\ \alpha_t \vec{p}_f(t-1) + (1-\alpha_t)\vec{p}(t), & \text{otherwise,} \end{cases} \tag{2}$$

where $\vec{p}(t) = \vec{p}_l(t)$ if only the local result $\vec{p}_l(t)$ for the current window is available at fusion time. Otherwise, we take the remote result as input, i.e., $\vec{p}(t) = \vec{p}_r(t)$, since the remote result is generally more accurate than the local one. The latter only happens when the remote model is at least as fast as the local model, i.e., $h_l \leq h_r$, and the network delay is negligible.

The state $\vec{p}_f(t-1)$ is periodically updated upon receiving a remote result. However, this update is *retrospective* since the result is most likely for a previous window due to the delay. Assume at time t we receive the remote result for window w_{t-N}. We update $\vec{p}_f(t-N), ..., \vec{p}_f(t-1)$ retrospectively to make sure the state information is properly reinforced by the remote result. For $i \in [t-N, t-1]$ we apply the following equation recursively.

$$\vec{p}_f(i) = \begin{cases} g(\vec{p}_l(i), \vec{p}_r(i)), & \text{if } i = t-N, \\ \alpha_i \vec{p}_f(i-1) + (1-\alpha_i)\vec{p}_l(i), & \text{otherwise.} \end{cases} \tag{3}$$

For window w_{t-N} we aggregate the local and remote results using a predefined aggregation function $g(\cdot)$. We use a weighted average function, one of the simple ensemble methods used in the area of model ensembling [45]. The weight parameter of the aggregation function $g(\cdot)$ is chosen empirically.

Our fusion method assumes the results from the DNN models to be a probability vector whose elements sum up to one. This assumption is realistic since, typically, modern classifiers like DNNs end with a *softmax* layer that generates a normalized probability vector.

B. Estimating Temporal Correlations

The FUSE and REINFORCE procedures in the fusion method are parameterized with parameter α_t which needs to be tuned according to the temporal correlation between windows w_{t-1}

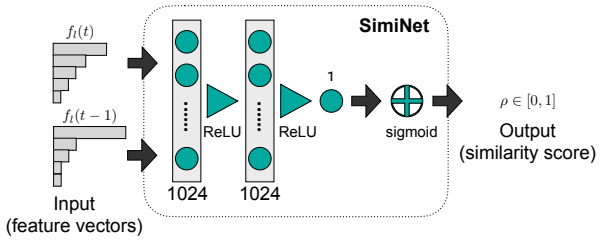

Fig. 6: SimiNet architecture. The input concatenates the pair of feature vectors $\vec{f_l}(t-1)$ and $\vec{f_l}(t)$ from two consecutive windows and is fed into two fully connected hidden layers each followed by a ReLU activation layer.

Algorithm 1 Calculation of α_t

$\vec{f_l}(t-1), \vec{f_l}(t)$ ▷ Local feature vectors
θ ▷ Threshold, default to 0.5
α_t ▷ Correlation parameter
β_t ▷ Cumulative remote influence

$\rho \leftarrow \text{SimiNet}(\vec{f_l}(t-1), \vec{f_l}(t))$
if $\beta_{t-1} < \theta$ **then** ▷ Apply moving average
    $\alpha_t \leftarrow 0.5$
else ▷ Apply weighted smoothing
    $\alpha_t \leftarrow \rho$
$\beta_t \leftarrow \beta_{t-1} \times \alpha_t$

and w_t. However, it is challenging to obtain the temporal correlation and decide whether a past remote result should be propagated through the current window, and if so, what portion it should take in the fused result at runtime. As we can observe from Figure 2, the temporal correlation is high when the context stays the same. Thus, the problem boils down to determining whether the current context is similar (or the same) to that of the previous window. From the context similarity score, we assign the parameter α_t accordingly.

Following this idea, we define a context similarity function which aims to capture the (dis)similarity of contexts based on information (e.g., the output probability vector or the intermediate feature vector) produced by the local model. The intuition behind this idea is that when the information produced by the local model for two consecutive windows has a high (low) correlation, it is very likely that the two windows belong to the same (different) context(s). Our argument is confirmed in Figure 8 (especially in the subplot titled SimiNet). If a high context similarity to the previous window is identified, we set a relatively large value for α_t. This way, the fused result $\vec{p_f}(t)$ will have more influence from the previous fused result $\vec{p_f}(t-1)$ which likely contains contribution from the remote model that is expected to be more accurate than the local one.

One simple approach to defining the context similarity function is to apply traditional similarity measures based on vector distance functions such as the Euclidean distance or Cosine similarity directly on the output of the model, i.e., the probability vector. However, our empirical studies reveal a large error with this approach. The main reason is two-fold: (a) The vector distance of the direct output of the local model is generally low due to the intra-class variations in the prediction. Such variations come from the relatively low accuracy of the local model. (b) The vector distance is not capable of capturing the subtle differences in the direct output of the local model. More specifically, the outputs of the local model for consecutive windows are similar to each other even when there is a clear context switch due to the large overlap in the windows that are neighbors to each other.

To overcome these limitations, we use a learning-based approach incorporating the following ideas: First, instead of the direct output of the model, we propose to capture the similarity on the feature vector, denoted by $f_l(t)$, which is extracted by the local model as intermediate information. Second, we propose to leverage a neural network, similar to the one used in [46], to learn the similarity. Recently, learning visual similarity using neural networks has been widely studied in fields like image retrieval [47] and person re-identification [48]. Our design for the neural network is called SimiNet and its structure is depicted in Figure 6. We use a four-layer perceptron which takes a pair of feature vectors that are generated by the local model on two consecutive windows, as input. We add a *sigmoid* function at the end of the network to produce the final similarity score between zero (totally different) and one (exactly the same). We train the SimiNet with pairs of inputs that contain labels zero or one with binary cross-entropy (BCE) loss function. We will elaborate on the training process in Section VI-A.

While the SimiNet captures the context similarity accurately, the fusion method should avoid taking a high portion of the previous fused result if this result does not carry much influence of the remote model. This can happen when the result from the remote model is delayed and by the time we calculate the fused result for a later window the remote result has not returned yet. To accommodate this situation, we introduce an indication variable, denoted by β_t, to track the accumulative influence of the remote model in the fused result, which is calculated based on the following equation.

$$\beta_t = \begin{cases} 0, & \text{if } t = 1, \\ 1, & \text{if } \vec{p_r}(t) \text{ is available,} \\ \beta_{t-1} \times \alpha_t, & \text{otherwise.} \end{cases} \tag{4}$$

Here, we reset the β_t to 1 every time we receive the remote result, meaning that the influence from the remote model is maximal. The influence then decreases with the advance of windows, incorporating the context similarity scores accumulatively. In the case that no remote result has been observed, i.e., β_t is smaller than a predefined threshold θ, we apply a standard moving average for the fusion where α_t is set to a predefined value (0.5 in our case). Note that β_t is also updated retrospectively upon calling the REINFORCE procedure. The details about α_t calculation are shown in Algorithm 1.

C. Handling Inaccurate Remote Results

While in general the accuracy of the remote model is higher than that of the local model, we cannot expect all results from the remote model to be correct. In the case that an incorrect result is returned by the remote model, it can be propagated through the following windows before a new result is received from the remote model, according to our fusion method design. This will lead to a cascading effect to the fusion performance, leading to low accuracy. One straightforward solution is to discard such incorrect remote results before we integrate them in the fusion method. However, identifying incorrect results (often known as uncertainty estimation in deep learning) is an open challenge [49].

To address this issue, we propose to leverage the confidence of the model prediction (the maximal value in the *softmax* probability vector generated by the model). More specifically, when we receive a remote result, we apply a predefined aggregation function $g(\cdot)$ defined in Equation (3) and set β_t to the confidence of the aggregated result. In this way, the fused result will be forwarded with a weight that equals the confidence of the aggregated result. Our intuition is that the confidence is relatively low when the (remote or local) model produces incorrect results. This assumption holds in general for calibrated neural networks [50].

V. Implementation

We implemented a prototype of Clownfish using commodity devices. Our software implementation is open-sourced.[1]

A. Hardware Setup

Similar to other work on edge video analytics [8], [20], we use an NVIDIA Jetson board as the edge node. In particular, we use a Jetson Xavier board [51] running Jetpack 4.3 (L4T 32.3.1) and equipped with eMMC 5.1 flash storage. The Jetson Xavier board includes a low-power mobile GPU and is suitable for massive edge deployment. We use a commodity server running CentOS-7.4 and equipped with Intel Xeon CPU E5-2630v3, 64 GB RAM, and NVIDIA RTX 2080Ti as the cloud node in our system. The edge and cloud nodes are connected using a local area network and we emulate different network conditions on this connection.

B. Software Implementation

The two DL model serving modules in Clownfish, i.e., Local and Remote, are implemented as standalone software components and their communication with the other modules in Clownfish is handled by gRPC [52]. This decoupling enables the flexibility of changing the DL models in different scenarios. We use Pytorch (v1.4.0) to perform the training and inference of the DL models used in the two modules.

The system is implemented as follows: We emulate a camera on the edge side by reading JPEG-formatted video frames pre-extracted from a video file with frame rate of 30 frames per second (FPS). We use the Python imaging library Pillow to read video frames from the flash storage. The eMMC flash on Jetson Xavier supports a high frame reading throughput (up to 1100 FPS) and thus, the frame reading latency is negligible. Once a frame window is formed, Clownfish encodes the window with PNG encoding, serializes the encoded window using protocol buffers, and makes a blocking gRPC call to the Local module running in a separate process. The Local module, upon receiving the gRPC call, decodes the frames from the window, processes them, and executes the local DL model on the GPU with Pytorch. For windows selected by the Filter module, Clownfish keeps a window queue and initializes a separate process to handle the windows in the queue by issuing non-blocking gRPC calls to the Remote module across the network. Note that overlapping frames across windows are processed once to avoid duplicate computation. The cloud side, upon receiving a frame window via a gRPC call, decodes the frames, processes them, and feeds them into the remote DL model running on the GPU with Pytorch.

Our fusion method is lightweight with respect to both computation and memory consumption. Memory consumption mainly comes from two parts: (1) the current window of a fixed number (i.e., window size) of frames and (2) the internal state information, including the result vectors and similarity scores for past d (the lag value) consecutive windows. We anticipate the memory consumption in both cases to be bounded and low, which is unlikely to become a bottleneck on resource-constraint edge devices.

VI. Performance Evaluation

In this section, we carry out extensive experiments to evaluate the performance of Clownfish. Our experiments aim to answer the following questions:

- **Q1** To what extent does our similarity-based fusion method improve the inference accuracy?
- **Q2** How do the filter interval and lag affect the inference accuracy and bandwidth reduction in Clownfish?
- **Q3** How does Clownfish perform under variable network conditions in real-world scenarios?
- **Q4** How well does Clownfish perform when compared with existing filtering-based approaches?

A. Experimental Setup

We choose the action recognition task in computer vision as a representative workload for Clownfish and all our experiments will be performed with this task.

Datasets. We evaluate our system on a popular action recognition benchmark called PKU-MMD [42], which contains long sequences of video frames with 51 action categories in total. The videos were captured using the Kinect v2 sensor in three camera views (left, right, and center). Each video runs for three to four minutes when played at 30 FPS, and contains multiple (around 20) action instances interleaved with backgrounds and performed by 66 subjects. The median value of the duration of action instances is 3.3 sec. The dataset contains two phases with large- and small-margin detection tasks, respectively. We specifically use the first phase which contains 1076 long

[1] https://github.com/vuhpdc/clownfish

videos. Following the cross-subject evaluation defined in [42], these videos are further split into training and testing sets with 994 videos by 57 subjects and 132 videos by 9 subjects, respectively, with a mix of different camera views.

DL models and training. For action recognition, we choose 3-dimensional convolutional neural networks (3D-CNNs) which have shown promising results in extracting spatio-temporal (i.e., appearance and motion) features from videos [53], [54]. More specifically, we adopt 3D residual networks of two different types, i.e., 3D ResNet-18 (smaller) with shortcut connections of type A and 3D ResNext-101 (larger) with shortcut connections of type B [55], to be used as the local and the remote model, respectively.

We take models that are pre-trained on the Kinetics dataset and fine-tune them on the PKU-MMD dataset for various spatial sizes (e.g., $224 \times 224 \times 3$) and temporal lengths (e.g., 16 frames) using the publicly available codebase[2] of 3D-ResNets [54]. We fine-tune the last two layers of the models, namely `conv5_x` and `fc`. For a fair comparison, we train all the models for the same number of iterations. It is non-trivial to choose the optimal values for the hyper-parameters in order to obtain the best performance of a model. Note that our goal is not to improve the accuracy of individual models but to verify the effectiveness of the fusion method. Therefore, we use default values for most of the hyper-parameters (e.g., SGD optimizer with learning rate 0.1, momentum 0.9, and weight decay 0.001) unless specified explicitly. In addition, models are trained only on those portions of a video that contain a class category (i.e., an action) in the ground truth. We use data augmentation techniques similar to the ones defined in the codebase, i.e., data normalization by mean, multi-scale random cropping from corners and center, random horizontal flip, and temporal random cropping for variable-length clips. During testing, we apply resizing, center cropping, and data normalization to the input data.

SimiNet training. We train SimiNet in Clownfish as a binary classification task where the `sigmoid` output of the network denotes the probability score of a default (in our case, similarity) class. We generate training samples of ordered pairs to the network as follows: For every semantically valid and variable-length context, denoted by C_t, in a training video, we take a fixed (e.g., twice the window size) number of frames immediately preceding this context, denoted by C_{t-1}. The ordered pairs (C_{t-1}, C_t) and (C_t, C_t) constitute two different training samples with ground truth labels zero (totally different) and one (exactly the same), respectively. For every element C_t in an ordered pair, the corresponding feature vector $\vec{f_l}(t)$ is extracted from the local model. Before passing input to the local model, we apply data augmentation techniques mentioned before. Note that the data augmentation technique named temporal random cropping, i.e., randomly selecting the start frame of an input, may generate different input combinations from an ordered pair. In addition, we use the SGD optimizer with a learning rate of 0.001, momentum 0.9, and weight decay 0.001. We train the model for 100 epochs with the batch size of 16.

Parameter settings. For all our experiments, we adopt the value of window size and window stride to be 16 and 4, respectively. We set the window size primarily based on the following three principles: (a) The input size of the temporal dimension of 3D networks is typically set to 16 frames [54], [56], [57], which is sufficient to capture most temporal patterns. (b) Contexts of lengths smaller than 16 frames (about half a second long) can be detected, assuming the result of the window is influenced by the majority of the frames in the window. (c) Smaller window sizes result in a lower amount of data transfer per window and faster feedbacks from the remote, which is critical in reducing bandwidth consumption and improving fusion performance, respectively. Similarly, we set the value for window stride to 4 (i.e., 75% overlap between consecutive windows) according to the speed of the local model (ResNet-18 in our case). The end-to-end inference latency of the local model is around 125 ms for window size 16. Therefore, the local model can process around 8 windows per second, when executed sequentially. In general, we expect the window stride to be as small as possible for better model accuracy due to denser sampling, but it is lower bounded by $\max\{1, \lceil h_s/h_l \rceil\}$ where h_s is the input frame rate and h_l is the throughput of the local model. We leave a detailed analysis of the impact of sliding window parameters on the accuracy and bandwidth as future work.

We use a spatial frame size of a window of $171 \times 128 \times 3$, which requires the network bandwidth to be around 8 Mbps after the PNG encoding for supporting 30 FPS. We set the value of filter interval n_w to be 6 unless specified explicitly, which can provide bandwidth reduction of up to 33% for the aforementioned sliding window parameters. In addition, we skip the first 8 frames of every context in accuracy calculation, because the label assigned to those frames could be from the previous context (i.e., the majority of frames in the window belongs to the previous context). For weighted aggregation function $g(\cdot)$ that aggregates the remote and local result, we use a weighted sum where we assign a weight of $2/3$ to the remote result to leave a weight of $1/3$ to the local result.

B. Performance Metrics

To measure the performance of our system, we use four different types of metrics described below.

Accuracy is defined as the fraction of correctly predicted frames (i.e., windows starting from each of these frames) out of the total number of frames that contain an action. We mark frame as true positive when the top-1 label matches the ground-truth label of the action context.

Weighted F1 score is a harmonic mean of the precision and the recall. This metric takes into account the label imbalance by considering the weight of each label, i.e., the number of true samples for each label.

Throughput is the total number of frames processed (for inference) per second by our system. This takes into account the amortized (frames that are skipped due to the

[2]https://github.com/kenshohara/3D-ResNets-PyTorch

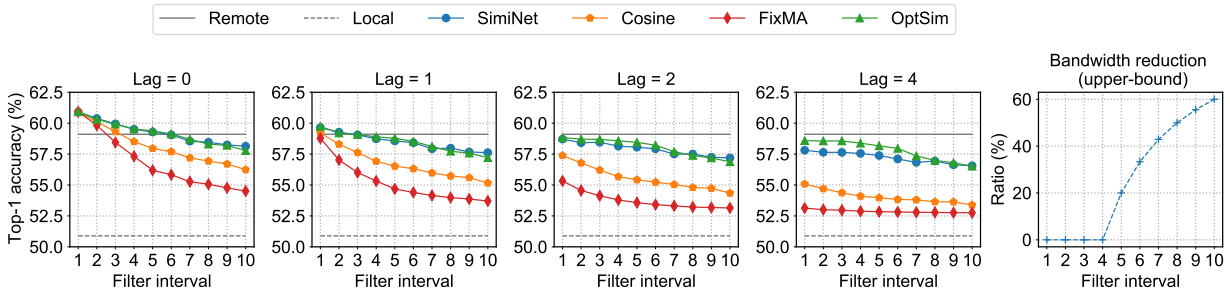

Fig. 7: Accuracy comparison between different fusion methods under varying filter intervals and lags and the bandwidth reduction achieved in the fusion methods under varying filter intervals.

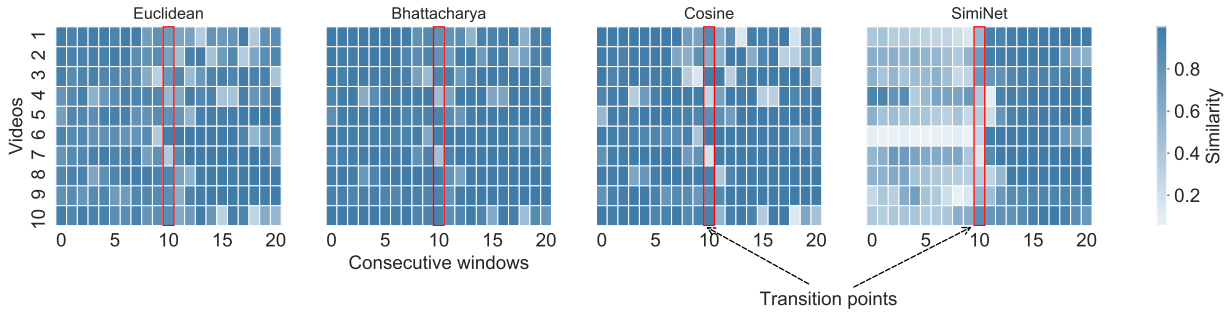

Fig. 8: Similarity scores generated by different similarity functions in 10 videos from the PKU-MMD dataset. The values for the Euclidean function represent the normalized values between 0 and 1 by assuming the maximum value is $\sqrt{2}$.

sliding window scheme have negligible inference time) end-to-end inference time which includes the encoding time of a window, network round-trip time to send the window and receive the inference result, window preprocessing, and the model execution time.

Bandwidth reduction is the amount of WAN traffic reduced by our system compared to the WAN traffic in the cloud-based system provided that both systems in comparison operate at the same throughput level.

C. Effectiveness of the SimiNet-based Fusion Method

We first evaluate the performance of SimiNet and examine the effectiveness of our SimiNet-based fusion method. In particular, we show how well SimiNet can capture the context similarity in videos and how SimiNet contributes to the overall inference performance of the fusion method in Clownfish under various scenarios.

Figure 8 shows the similarity scores captured by SimiNet in comparison with several other popular functions for similarity characterization on 10 videos from the PKU-MMD dataset. Each of the videos contains a transition point in the middle where the windows before the transition point contain no actions (except the first 5 windows in the 4th video) and the windows after the transition point contain a valid action. As we can see, SimiNet distinguishes the transition point clearly by generating lower similarity scores around the transition point and higher similarity scores for the windows with the same action as expected. The other similarity functions are only able to detect the transition points in very few videos. This comparison demonstrates that our learning-based SimiNet is more effective in characterizing the context similarity in videos than existing functions.

We now evaluate the accuracy of our SimiNet-based fusion method under different filter intervals n_w and lags d defined as the delay of receiving the remote feedback measured in number of windows. The case with lag $d = 0$ stands for an ideal case where there is no delay in receiving the remote results. We compare our SimiNet-based method with three other fusion methods namely Cosine, FixMA, and OptSim. The Cosine method assigns parameter α_t in Equation (1) using the Cosine similarity score. The FixMA method sets the parameter α_t to a constant 0.5. The OptSim method represents the ideal case where parameter α_t is set according to the similarity ground truth, i.e., 1 for the same action and 0 otherwise. Note that OptSim can only be achieved in theory since the ground truth is unknown beforehand in practice.

Figure 7 depicts the accuracy comparison results under filter intervals $n_w \in [1, 10]$ and lags $d = 0, 1, 2, 4$. It can be observed that **our SimiNet-based fusion method performs close to the remote model and the accuracy gap is around** 2% **in almost all cases (Q1)**. The SimiNet is also effective as the overall accuracy with SimiNet is close to that with OptSim. Our SimiNet-based fusion method outperforms all other methods by a large margin most of the time. In general, **the performance of our method is stable even under large filter intervals and large lags (Q2).**

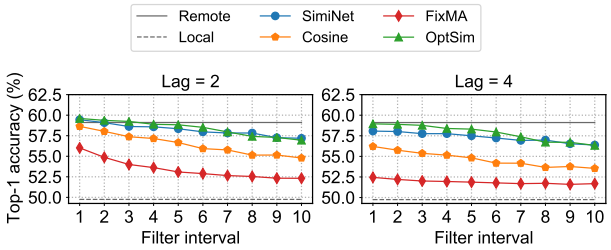

Fig. 9: Accuracy achieved by our proposed fusion method with 3D MobileNet v1 as the local model under varying filter intervals and lags.

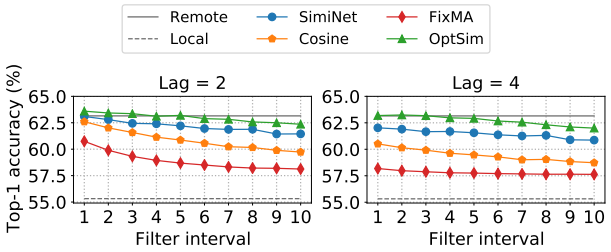

Fig. 10: Accuracy achieved by our proposed fusion method on cross-view split.

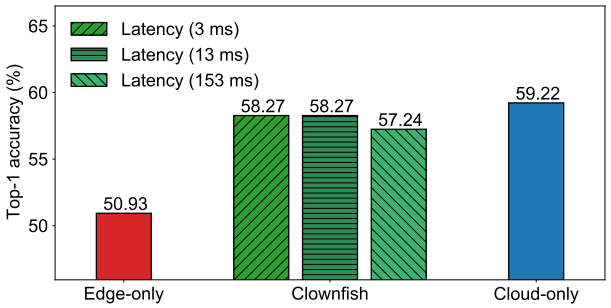

Fig. 11: Accuracy achieved by Clownfish under different network latencies compared with the edge- and cloud-only solutions.

The rightmost plot in Figure 7 depicts the bandwidth reduction when facing different filter intervals in our method. Note that the bandwidth reduction can only be observed with filter intervals $n_w > 4$, i.e., when the filter interval is greater than the ratio between the window size and the window stride. Although the overall accuracy drops gradually with the increase of the filter interval as expected (due to less remote feedbacks), the bandwidth reduction increases substantially. For example, for lag $d = 2$, the accuracy drops from 57.93% to 57.50% when we increase the filter interval from 6 to 7, while the bandwidth reduction increases from 33.33% to 42.86%. Overall, **our fusion method can achieve substantial bandwidth reduction with a limited penalty on the inference accuracy (Q2)**, being less dependent on the available network bandwidth.

We observe that the accuracy of the SimiNet-based fusion method is better than the accuracy of the remote model in some cases, especially when the values for n_w and d are low. We attribute this gain to the weighted moving average which aggregates results across windows. This is confirmed by the fact that the moving average method can also help improve the accuracy of the local and remote models. We will discuss it in more detail in Section VI-F.

Additionally, we report the applicability of SimiNet architecture to other types of local models. To this end, we fine-tune 3D MobileNet v1 (pre-trained on the Kinetics dataset) on PKU-MMD dataset following the training strategy proposed in [58]. We then train SimiNet on the feature vectors generated by the 3D MobileNet, following the training strategy stated in Section VI-A. Figure 9 depicts the accuracy comparison with varying filter intervals and under lags $d = 2, 4$. As we can see that the observation presented in Section VI-C, i.e., SimiNet-based fusion method outperforms other methods by a large margin and is close to the OptSim, still holds. This confirms the generality of SimiNet.

To further test the effectiveness of our approach, we evaluated the fusion method also with a different dataset split method called cross-view split defined in [42]. In cross-view split, the PKU-MMD dataset is split into 717 training videos with left and right camera views, with 359 testing videos with center view. According to [42], cross-view evaluation aims to test the robustness of the models in terms of transformation (e.g., translation, rotation). In contrast, cross-subject evaluation aims to test the intra-class variation among different actors. Figure 10 depicts the accuracy with varying filter interval values following the same training strategy and models used in the cross-subject evaluation. Similar to the above observations, our fusion method outperforms other methods by a significant margin. Note that the accuracy of individual models is higher in cross-view evaluation, consistent with the evaluation results mentioned in [42].

TABLE II: Weighted F1 scores under varying network conditions (left: varying latency, right: varying bandwidth).

Solution	**F1 (%)**	**Solution**	**F1 (%)**
Edge-only	50.58	Edge-only	50.36
Clownfish (3 ms)	58.33	Clownfish (5 Mbps)	56.15
Clownfish (13 ms)	58.34	Clownfish (7.5 Mbps)	56.51
Clownfish (153 ms)	57.18	Clownfish (No shaping)	58.59
Cloud-only	59.68	Cloud-only	59.13

D. Impact of Network Latency

We evaluate the performance of Clownfish in real-world scenarios with varying network latency conditions. In particular, we compare Clownfish with the edge- and cloud-only solutions in terms of accuracy (together with weighted F1 score) and throughput. The edge-only solution runs the ResNet-18 model, while the cloud-only solutions runs the ResNext-101 model. For the experiments with varying latency conditions, we choose to use all 132 testing videos in the dataset to report the accuracy and the weighted F1 score.

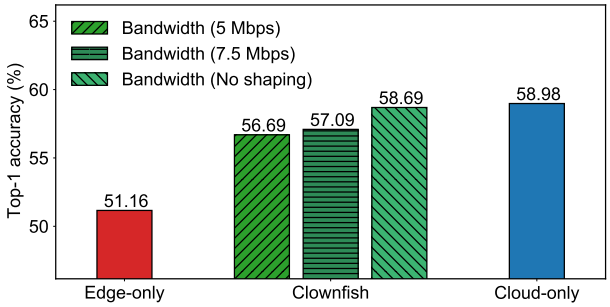

Fig. 12: Accuracy achieved by Clownfish under different bandwidth shaping conditions compared with the edge- and cloud-only solutions.

Figure 11 shows the accuracy achieved by Clownfish under varying network latency conditions. We choose three realistic network latency conditions, i.e., 3 ms, 13 ms, and 153 ms, which are deduced from the average latencies we measured for a local-area (within Amsterdam), inner-continent (Amsterdam-Frankfurt), and inter-continent (Amsterdam-W. California) connection from the Jetson board to an Amazon cloud server, respectively. We emulate these latency conditions on our prototype using the `netem` tool. We put no limits on the network bandwidth and ensure the network bandwidth is not a bottleneck. It can be observed that the accuracy of Clownfish under latencies of 3 ms and 13 ms are almost the same, and more than 96% of remote results are returned within the lag $d = 1$. However, with network latency of 153 ms, the network latency contributes to the end-to-end delay significantly, leading to 5.2%, 53.2%, and 39.5% of remote results to be returned with lags of 2, 3, and 4, respectively. In general, Clownfish is able to maintain a stable accuracy performance and performs close to the cloud-only solution in all cases. The weighted F1-scores shown in Table II (left) reveal similar trends. Overall, **network latency has a negligible impact on the achieved accuracy of Clownfish (Q3).**

E. Impact of Network Bandwidth

We explore the performance of Clownfish under varying network bandwidth conditions. To emulate the bandwidth variation, we follow the traffic shaping experiment (including the bandwidth control values) similar to the one used in AWStream [39] that targets the wide area. We use the Linux `tc` utility to control the uplink bandwidth from edge to cloud. For the experiment, we pick 12 videos from the testing videos in the dataset, with equal numbers of videos in each camera view category, namely left, right, or centre. We run experiments in the following order: the first three videos with no bandwidth shaping (A), the next three videos with 7.5 Mbps bandwidth limit (B), the next three videos with 5 Mbps limit (C), and the last three videos again with no shaping (A). We collect throughput statistics every 5 sec. The x-axis in Figure 13 depicts the approximate time sequence.

Figure 12 depicts the accuracy of Clownfish on the 12 videos under different network bandwidth conditions. As

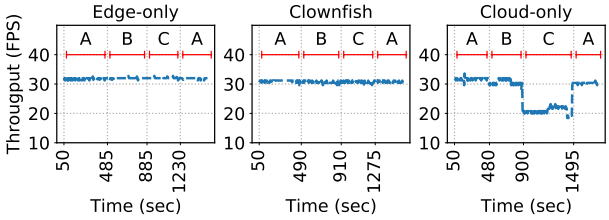

Fig. 13: System throughput (FPS) under different bandwidth conditions (A: no shaping, B: 7.5 Mbps, C: 5 Mbps).

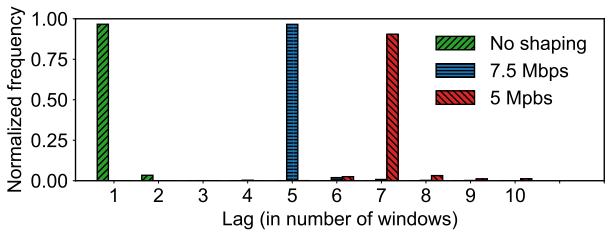

Fig. 14: Distribution of lags of remote feedbacks under varying network bandwidth conditions.

expected, Clownfish always performs close to the cloud-only solution and outperforms the edge-only solution to a large extent. However, we observe a slight accuracy decrease with the decrease of the network bandwidth. The accuracy degradation is due to the increase of lags of the remote feedbacks, as clearly shown in Figure 14. This lag increase is caused by the longer time required to transmit the original frame data under lower bandwidth.

Figure 13 depicts the throughput results of Clownfish in comparison with the edge- and cloud-only solutions. The throughput of the edge-only solution remains almost constant (around 31 FPS) since the network is not involved. The cloud-only solution, however, suffers from poor throughputs when the network bandwidth is lower than the required bandwidth (7.5-8 Mbps in the considered scenario). In particular, the throughput drops to two-third of the expected throughput, i.e., 30 FPS when the bandwidth is limited to 5 Mbps. This confirms that the cloud-only solution will not be able to maintain stable throughputs under varying network bandwidths, which are typical on the WAN. Despite the bandwidth variation, Clownfish always maintain a stable throughput thanks to hybrid design, where the throughput is dictated by the local model while the remote model is only used for accuracy improvements. Overall, **Clownfish is able to maintain stable throughputs as the edge-only solution and achieve accuracies as high as the cloud-only solution under varying network bandwidth conditions (Q3).**

F. The Power of Leveraging Temporal Correlations

To further demonstrate the power of leveraging temporal correlations as done in Clownfish, we apply the fixed moving average (FixMA) following Equation (1), with parameter α_t set to 0.5, on the results obtained by the local and remote

TABLE III: Accuracy comparison with fixed moving average applied on the edge- and cloud-only solutions.

Solutions	Accuracy (%)	F1 (%)
Edge-only	50.93	50.58
Edge-only (+ FixMA)	52.54	52.11
Clownfish (13 ms)	58.27	58.34
Cloud-only	59.22	59.68
Cloud-only (+ FixMA)	61.50	61.99

models. Table III shows that both the local and remote models benefit from FixMA where the accuracy is improved by around 2%. Even though the advantage of Clownfish against the edge-only solution is obvious, the accuracy of Clownfish remains comparable to that of the cloud-only solution with FixMA.

G. Comparison with Existing Approaches

We compare Clownfish with the existing filtering-based approach, where irrelevant frames are identified and discarded before being sent to the cloud for processing. This approach has been used for object recognition in EarlyDiscard [40] and FilterForward [59].

To adapt the filtering-based approach for action recognition, we train a 3D version of MobileNet V1 to discard irrelevant windows (instead of frames in EarlyDiscard). This is because discarding frames using 2D MobileNet V1 does not consider the motion features which are important for action recognition. We fine-tune the last classifier layer of 3D-MobileNet V1 (pre-trained on the Kinetics dataset) similar to the training process proposed in [58] but for the binary classification task. We conduct our experiments on the 12 videos used in the bandwidth shaping experiments. We set the cutoff threshold of the binary classification to 0.5 as done in EarlyDiscard to achieve high recall (over 0.9).

With the above setup, the filtering-based approach achieves an overall accuracy of 55.48%, which is lower than the cloud-only solution (58.98%) but is considerably higher than the edge-only solution (51.16%). The fraction of frames sent to the cloud per video is between 44.10% to 76.31% where the fraction of relevant frames (with actions) is between 30.15% and 53.40%. Compared with Clownfish, the accuracy of the filter-based approach is still worse.

Figure 15 shows the throughput performance of the filtering-based approach under varying network bandwidth conditions. We consider two scenarios with no bandwidth shaping and shaping with a limit of 5 Mbps and perform the experiments on the video named 0316-R, which has around 64% of relevant frames. Due to the performance limit of 3D MobileNet (with a latency of around 148 ms), the theoretical throughput of the filtering-based approach is around 26 FPS, which is lower than the expected throughput of 30 FPS. As we can see from Figure 15 that when no bandwidth limits are imposed, the filtering-based approach can maintain a relatively stable throughput close to the theoretical optimal. However, when the bandwidth is limited, the throughput drops to 20 FPS most of the time due to the large latency in transmitting the relevant frames, which are still significant in number.

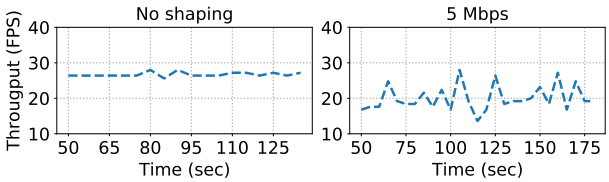

Fig. 15: Throughput of the filtering-based approach under varying network bandwidth conditions.

Overall, **Clownfish outperforms the filtering-based approach with respect to both accuracy and throughput (Q4).** Nevertheless, filtering-based approach can be made complementary to Clownfish where irrelevant frames (or windows) are identified and discarded to further reduce bandwidth consumption in Clownfish. We leave such a mixed approach for future work.

VII. Related Work

A. Computation Offloading for Video Stream Analytics

Computation offloading has been extensively explored in the context of mobile computing [60], [61]. The focus is mainly on providing runtime support for offloading parts of monolithic mobile applications to a cloud or a nearby server with the objective of minimizing application execution time or mobile energy consumption. Recently, similar ideas have also been applied to video stream analytics workloads [6]–[8], [40], [62]–[65]. Modern video stream analytics workflows typically leverage complex DNN models for computer vision tasks such as object or action recognition, and the DNN models can be offloaded to and edge/cloud server partially or as a whole.

Most solutions to computation offloading for video stream analytics focus on network bandwidth or latency issues. JetStream addresses the bandwidth variation issue through data quality adaptation [36]. AWStream pushes this solution further and allows the adaptation of the DNN models to achieve the best tradeoff between analytics accuracy and latency in video stream analytics [7]. Filtering-based approaches such as EarlyDiscard [40] and FilterForward [59] employ a filtering technique at the edge to discard the context-irrelevant frames and send only the relevant frames to the cloud for processing. CloudSeg employs the super-resolution technique where low-resolution frames are sent to the cloud, which are reconstructed into high-resolution ones for further analytics [64]. Similar to Clownfish, Glimpse [62] adopts a hybrid design where the object recognition task is offloaded to the (edge) cloud. Glimpse proposes a local tracking approach based on light-weight optical flow calculation to combat the network latency issue. Clownfish takes a fundamentally different approach where the local edge device is used for actual analytics tasks while the cloud is used for result reinforcements. In addition, Clownfish is generally applicable to a variety of analytics tasks with temporal correlations, while Glimpse only targets object recognition.

Another line of research proposes to split the DNN model for video stream analytics and execute partially on the edge and on the cloud [65]–[67]. While achieving lower latency in general, these methods only work for certain DNN models and still struggle to provide real-time performance especially when the network bandwidth is limited [66].

B. DNN Optimizations for Edge Deployment

Recently, there has been a significant amount of work on making DNN models more compact, generally know as model compression, to enable their deployment on resource constrained edge devices. Popular model compression techniques such as pruning, quantization, and knowledge distillation [23]–[27] have shown encouraging results on simple tasks such as image classification. However, such techniques generally suffer from accuracy degradation. On the other hand, models are becoming increasingly larger. For example, the recent NLP model GPT-3 has around 175 billion parameters making it impossible to deploy it on edge devices even after the aforementioned techniques. With our hybrid design, one can deploy the optimized and specialized DNNs produced using model compression techniques and periodically take help from the large original models. Leite et al. [68] proposed an efficient solution for activity recognition on mobile devices by using an LSTM-based architecture to model inter-window temporal context and avoid overlapping computation in sliding window schemes. This approach is complementary to our work.

VIII. Discussion and Future Work

In this section, we discuss some of the challenges, optimizations, and future work on Clownfish.

Our idea of aggregating the remote and local results is inspired by the model ensemble approach, where the analytics results of multiple models are combined to give better and more robust results than the individual models. We used a linear weighted average function as our aggregation function, and empirically selected the value of the weight parameter. However, there are many methods, including exhaustive grid search, estimation algorithms such as linear regression, to estimate the optimal weights. It is also worth exploring the more advanced aggregation methods such as boosting or stacking [45].

Currently, the Clownfish evaluation uses a fixed value for the filter interval in the fusion method. Given the application bandwidth requirement for a window, available network bandwidth, and the end-to-end network latency (i.e., lag), the fixed value can be determined by examining the trade-off curve between accuracy and bandwidth reduction (similar to Figure 7) plotted using training or holdout validation dataset for different filter intervals. However, such a fixed value might be suboptimal in variable network conditions. An adaptive approach where the filter interval is set based on the current network condition will help improve the overall accuracy of the fusion method.

Clownfish is only able to employ temporal correlations within the same context (e.g., same action). At context transition points (action-to-action or background-to-action), our framework will generate a low correlation (similarity) score, which significantly reduces the influence of the wrong remote result (that belongs to a previous context) on the current context. However, it would be interesting to see if inter-context correlations can also be explored and incorporated.

Model optimization or compression techniques may degrade the accuracy of the DL model depending upon the model architecture, hardware platform, compression method, and compression ratio [28]. In such a scenario, employing Clownfish and aggressive optimization methods will further improve the accuracy of the optimized model at a minimal cost of network bandwidth.

IX. Conclusion

We presented Clownfish, a hybrid real-time video analytics framework that achieves edge-and-cloud symbiosis to reap the benefits of both the edge (fast response) and the cloud (high accuracy). Clownfish exploits the temporal correlations in the video content and uses a learning-based approach to characterize the temporal correlation with a similarity score metric. Clownfish then employs an effective fusion method based on the exponential smoothing technique for fusing the analytics results from the optimized model running on the edge and the intermittent and possibly delayed results from the complete model running in the cloud. Our evaluation confirms that Clownfish can always operate in real time as the edge-only solution and achieves comparable accuracy with the cloud-only solution even in high network latency and varying network bandwidth conditions.

X. Acknowledgement

We would like to thank the anonymous reviewers and our shepherd Mi Zhang for their valuable comments and suggestions. We would also like to thank Benno Kruit and Leonardos Pantiskas for proofreading the paper. This work is part of the Efficient Deep Learning (EDL) programme (grant number P16-25), financed by the Dutch Research Council (NWO). Part of the work was carried out on the NWO-funded DAS-5 cluster.

References

[1] H. Zhang, G. Ananthanarayanan, P. Bodík, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017, pp. 377–392.

[2] G. Ananthanarayanan, V. Bahl, L. P. Cox, A. Crown, S. Nogbahi, and Y. Shu, "Video analytics - killer app for edge computing," in *ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2019, pp. 695–696.

[3] C. Hung, G. Ananthanarayanan, P. Bodík, L. Golubchik, M. Yu, P. Bahl, and M. Philipose, "Videoedge: Processing camera streams using hierarchical clusters," in *IEEE/ACM Symposium on Edge Computing (SEC)*, 2018, pp. 115–131.

[4] G. Grassi, K. Jamieson, P. Bahl, and G. Pau, "Parkmaster: an in-vehicle, edge-based video analytics service for detecting open parking spaces in urban environments," in *ACM/IEEE Symposium on Edge Computing (SEC)*, 2017, pp. 16:1–16:14.

[5] S. Jain, V. Nguyen, M. Gruteser, and P. Bahl, "Panoptes: servicing multiple applications simultaneously using steerable cameras," in *ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2017, pp. 119–130.

[6] J. Jiang, G. Ananthanarayanan, P. Bodík, S. Sen, and I. Stoica, "Chameleon: scalable adaptation of video analytics," in *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2018, pp. 253–266.

[7] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, and E. A. Lee, "Awstream: adaptive wide-area streaming analytics," in *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2018, pp. 236–252.

[8] J. Wang, Z. Feng, Z. Chen, S. A. George, M. Bala, P. Pillai, S. Yang, and M. Satyanarayanan, "Bandwidth-efficient live video analytics for drones via edge computing," in *IEEE/ACM Symposium on Edge Computing (SEC)*, 2018, pp. 159–173.

[9] S. Y. Jang, Y. Lee, B. Shin, and D. Lee, "Application-aware iot camera virtualization for video analytics edge computing," in *IEEE/ACM Symposium on Edge Computing (SEC)*, 2018, pp. 132–144.

[10] H. Sun, X. Liang, and W. Shi, "Vu: video usefulness and its application in large-scale video surveillance systems: an early experience," in *IEEE Workshop on Smart Internet of Things (SmartIoT@SEC)*, 2017, pp. 6:1–6:6.

[11] R. Poddar, G. Ananthanarayanan, S. Setty, S. Volos, and R. A. Popa, "Visor: Privacy-preserving video analytics as a cloud service," *CoRR*, vol. abs/2006.09628, 2020.

[12] T. Abdullah, A. Anjum, M. F. Tariq, Y. Baltaci, and N. Antonopoulos, "Traffic monitoring using video analytics in clouds," in *IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*, 2014, pp. 39–48.

[13] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *IEEE Computer*, vol. 50, no. 10, pp. 58–67, 2017.

[14] B. Luo, S. Tan, Z. Yu, and W. Shi, "Edgebox: Live edge video analytics for near real-time event detection," in *IEEE/ACM Symposium on Edge Computing (SEC)*, 2018, pp. 347–348.

[15] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2017, pp. 4278–4284.

[16] V. Veeriah, N. Zhuang, and G. Qi, "Differential recurrent neural networks for action recognition," in *IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 4041–4049.

[17] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *Annual Conference on Neural Information Processing Systems (NIPS)*, 2014, pp. 568–576.

[18] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 1, pp. 221–231, 2013.

[19] H. Shen, L. Chen, Y. Jin, L. Zhao, B. Kong, M. Philipose, A. Krishnamurthy, and R. Sundaram, "Nexus: a GPU cluster engine for accelerating dnn-based video analysis," in *ACM Symposium on Operating Systems Principles (SOSP)*, 2019, pp. 322–337.

[20] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *ACM Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2019, pp. 25:1–25:16.

[21] S. Biswas, J. C. Bicket, E. Wong, R. Musaloiu-E, A. Bhartia, and D. Aguayo, "Large-scale measurements of wireless network behavior," in *ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, 2015, pp. 153–165.

[22] A. Nikravesh, D. R. Choffnes, E. Katz-Bassett, Z. M. Mao, and M. Welsh, "Mobile network performance from user devices: A longitudinal, multidimensional analysis," in *International Conference on Passive and Active Measurement (PAM)*, vol. 8362, 2014, pp. 12–22.

[23] X. Zhang, H. Lu, C. Hao, J. Li, B. Cheng, Y. Li, K. Rupnow, J. Xiong, T. S. Huang, H. Shi, W. W. Hwu, and D. Chen, "Skynet: a hardware-efficient method for object detection and tracking on embedded systems," in *Conference on Machine Learning and Systems (MLSys)*, 2020.

[24] M. Rusci, A. Capotondi, and L. Benini, "Memory-driven mixed low precision quantization for enabling deep network inference on microcontrollers," in *Conference on Machine Learning and Systems (MLSys)*, 2020.

[25] Y. Zhou, S. Moosavi-Dezfooli, N. Cheung, and P. Frossard, "Adaptive quantization for deep neural network," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2018, pp. 4596–4604.

[26] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, "A systematic DNN weight pruning framework using alternating direction method of multipliers," in *European Conference on Computer Vision (ECCV)*, vol. 11212, 2018, pp. 191–207.

[27] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *CoRR*, vol. abs/1503.02531, 2015.

[28] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Signal Process. Mag.*, vol. 35, no. 1, pp. 126–136, 2018.

[29] S. Y. Jang, Y. Lee, B. Shin, and D. Lee, "Application-aware iot camera virtualization for video analytics edge computing," in *IEEE/ACM Symposium on Edge Computing (SEC)*, 2018, pp. 132–144.

[30] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *IEEE Computer*, vol. 50, no. 10, pp. 58–67, 2017.

[31] Y. Lin, F. Lv, S. Zhu, M. Yang, T. Cour, K. Yu, L. Cao, and T. S. Huang, "Large-scale image classification: Fast feature extraction and SVM training," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011, pp. 1689–1696.

[32] D. C. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012, pp. 3642–3649.

[33] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[34] B. Zhang, L. Wang, Z. Wang, Y. Qiao, and H. Wang, "Real-time action recognition with enhanced motion vector cnns," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2718–2726.

[35] TFLite hosted models, https://www.tensorflow.org/lite/guide/hosted_models, (Accessed on June 16, 2020).

[36] A. Rabkin, M. Arye, S. Sen, V. S. Pai, and M. J. Freedman, "Aggregation and degradation in jetstream: Streaming analytics in the wide area," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2014, pp. 275–288.

[37] "Cisco annual internet report (2018-2023) white paper," https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[38] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, "Gaia: Geo-distributed machine learning approaching LAN speeds," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017, pp. 629–647.

[39] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, and E. A. Lee, "Awstream: adaptive wide-area streaming analytics," in *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2018, pp. 236–252.

[40] J. Wang, Z. Feng, Z. Chen, S. A. George, M. Bala, P. Pillai, S. Yang, and M. Satyanarayanan, "Bandwidth-efficient live video analytics for drones via edge computing," in *IEEE/ACM Symposium on Edge Computing (SEC)*, 2018, pp. 159–173.

[41] M. Satyanarayanan, "The emergence of edge computing," *IEEE Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[42] L. Chunhui, H. Yueyu, L. Yanghao, S. Sijie, and L. Jiaying, "Pku-mmd: A large scale benchmark for continuous multi-modal human action understanding," *arXiv preprint arXiv:1703.07475*, 2017.

[43] R. G. Brown, "Smoothing, forecasting and prediction of discrete time series," 1962.

[44] S. Bianco, R. Cadène, L. Celona, and P. Napoletano, "Benchmark analysis of representative deep neural network architectures," *IEEE Access*, vol. 6, pp. 64 270–64 277, 2018.

[45] Z.-H. Zhou, "Ensemble methods: foundations and algorithms," 2012.

[46] N. Garcia and G. Vogiatzis, "Learning non-metric visual similarity for image retrieval," *Image Vis. Comput.*, vol. 82, pp. 18–25, 2019.

[47] E. Hoffer and N. Ailon, "Deep metric learning using triplet network," in *Similarity-Based Pattern Recognition - Third International Workshop, SIMBAD*, ser. Lecture Notes in Computer Science, vol. 9370, 2015, pp. 84–92.

[48] D. Yi, Z. Lei, S. Liao, and S. Z. Li, "Deep metric learning for person re-identification," in *22nd International Conference on Pattern Recognition, ICPR*, 2014, pp. 34–39.

[49] Y. Gal, "Uncertainty in deep learning," Ph.D. dissertation, University of Cambridge, 2016.

[50] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *ICML*, 2017, pp. 1321–1330.

[51] NVIDIA AGX Jetson Xavier, https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/, (Accessed on July 1, 2020).

[52] gRPC, https://grpc.io, (Accessed on June 1, 2020).

[53] J. Carreira and A. Zisserman, "Quo vadis, action recognition? A new model and the kinetics dataset," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2017, pp. 4724–4733.

[54] K. Hara, H. Kataoka, and Y. Satoh, "Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet?" in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[55] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[56] D. Tran, L. D. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *ICCV*, 2015, pp. 4489–4497.

[57] Z. Qiu, T. Yao, and T. Mei, "Learning spatio-temporal representation with pseudo-3d residual networks," in *ICCV*, 2017, pp. 5534–5542.

[58] O. Köpüklü, N. Kose, A. Gunduz, and G. Rigoll, "Resource efficient 3d convolutional neural networks," in *ICCV Workshops*, 2019, pp. 1910–1919.

[59] C. Canel, T. Kim, G. Zhou, C. Li, H. Lim, D. G. Andersen, M. Kaminsky, and S. R. Dulloor, "Scaling video analytics on constrained edge nodes," in *SysML*, 2019.

[60] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: making smartphones last longer with code offload," in *ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2010, pp. 49–62.

[61] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *ACM European Conference on Computer Systems (EuroSys)*, 2011, pp. 301–314.

[62] T. Y. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices," in *SenSys*. ACM, 2015, pp. 155–168.

[63] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "LAVEA: latency-aware video analytics on edge computing platform," in *ACM/IEEE Symposium on Edge Computing (SEC)*, 2017, pp. 15:1–15:13.

[64] Y. Wang, W. Wang, J. Zhang, J. Jiang, and K. Chen, "Bridging the edge-cloud barrier for real-time advanced vision analytics," in *11th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud*, 2019.

[65] J. Emmons, S. Fouladi, G. Ananthanarayanan, S. Venkataraman, S. Savarese, and K. Winstein, "Cracking open the DNN black-box: Video analytics with dnns across the camera-cloud boundary," in *ACM Workshop on Hot Topics in Video Analytics and Intelligent Edges (HotEdgeVideo)*, 2019, pp. 27–32.

[66] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. N. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2017, pp. 615–629.

[67] P. M. Grulich and F. Nawab, "Collaborative edge and cloud neural networks for real-time video processing," in *VLDB Endowment*, 2018.

[68] C. F. Souza Leite and Y. Xiao, "Improving resource efficiency of deep activity recognition via redundancy reduction," in *ACM International Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2020.