ShadowTutor: Distributed Partial Distillation for Mobile Video DNN Inference

Jae-Won Chung*
Seoul National University
Seoul, South Korea
jaywonchung@snu.ac.kr

Jae-Yun Kim
Seoul National University
Seoul, South Korea
jaeykim@altair.snu.ac.kr

Soo-Mook Moon*
Seoul National University
Seoul, South Korea
smoon@snu.ac.kr

ABSTRACT

Following the recent success of deep neural networks (DNN) on video computer vision tasks, performing DNN inferences on videos that originate from mobile devices has gained practical significance. As such, previous approaches developed methods to offload DNN inference computations for images to cloud servers to manage the resource constraints of mobile devices. However, when it comes to video data, communicating information of every frame consumes excessive network bandwidth and renders the entire system susceptible to adverse network conditions such as congestion. Thus, in this work, we seek to exploit the *temporal coherence* between nearby frames of a video stream to mitigate network pressure. That is, we propose *ShadowTutor*, a distributed video DNN inference framework that reduces the number of network transmissions through intermittent *knowledge distillation* to a student model. Moreover, we update only a subset of the student's parameters, which we call *partial distillation*, to reduce the data size of each network transmission. Specifically, the server runs a large and general *teacher* model, and the mobile device only runs an extremely small but specialized *student* model. On sparsely selected *key frames*, the server partially trains the student model by targeting the teacher's response and sends the updated part to the mobile device. We investigate the effectiveness of ShadowTutor with HD video semantic segmentation. Evaluations show that network data transfer is reduced by 95% on average. Moreover, the throughput of the system is improved by over three times and shows robustness to changes in network bandwidth.

CCS CONCEPTS

• **Human-centered computing** → **Mobile computing**; • **Computing methodologies** → **Distributed artificial intelligence**; **Distributed computing methodologies**.

KEYWORDS

Deep neural networks, distributed inference, knowledge distillation, video semantic segmentation

*corresponding authors

ICPP '20, August 17–20, 2020, Edmonton, AB, Canada

ACM ISBN 978-1-4503-8816-0/20/08…$15.00
https://doi.org/10.1145/3404397.3404404

ACM Reference Format:
Jae-Won Chung, Jae-Yun Kim, and Soo-Mook Moon. 2020. ShadowTutor: Distributed Partial Distillation for Mobile Video DNN Inference. In *49th International Conference on Parallel Processing - ICPP (ICPP '20), August 17–20, 2020, Edmonton, AB, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3404397.3404404

1 INTRODUCTION

Deep learning approaches have proved to be extremely powerful in fields such as computer vision [13], natural language processing [29], speech recognition [4], and sequential recommendation [14]. Due to their applicability to various services, such DNN models sparked interest for performing DNN inference directly on data generated on mobile devices.

Video is one of the most important among such data due to its numerous practical applications. Autonomous vehicles perform road segmentation to traverse the street and detect obstacles [27]. CCTV cameras used for home security can notice movements and detect objects using DNNs [16]. Mobile selfie apps segment humans and the rest to change the background or simulate background blurring effects [3]. Thus, video DNN inferences on comparatively weak devices have become a core mechanism for delivering various services and technologies.

While running a large model directly on-device may yield high accuracy, execution latency and memory consumption can quickly become unacceptable for resource-constrained mobile devices. Thus, to lift the mobile device of the unwieldy inference workload, several approaches offload DNN computations to cloud servers [8, 12, 17]. In this scheme, the mobile device sends video frames to the cloud server and retrieves inference results. However, these approaches are limited in that information of every frame must be communicated between the server and the client. This heavily pressures network traffic, which is a view shared with [1]. Moreover, the entire system becomes vulnerable to adverse network conditions such as reduction in available network bandwidth due to congestion.

To this end, we propose *ShadowTutor*, a general video DNN inference framework. Here, a very small *pre-trained* student model runs on a mobile device for test time inference. However, due to its size, the student lacks generalization power to excel on any given input scene even if it has undergone "public education". Thus, to enhance performance specifically on the video at hand, the student receives "shadow education" from a large teacher model on the server. Especially, the student receives training only on a fraction of all frames in the video, which frames we call *key frames*.

The student model on the mobile device processes video frames one by one in an online manner. On sparse *key frames*, the mobile device sends the frame to the server. Upon key frame receive, the server updates a copy of the student by considering the teacher's

inference result as the key frame's label, and returns the updated student weights to the mobile device. In effect, the student is specialized to the current video stream by distilling knowledge from the teacher during test time. Now, the student model can accurately perform inference on-device without the teacher's help for *non-key frames*, thanks to the *temporal coherence* between the key frame and the non-key frames after it. That is, non-key frames are likely to share characteristics such as background, ambience, overall texture, speed of movement, and the objects present with a nearby key frame, thus allowing the student trained on that key frame to accurately process non-key frames that come after it. Moreover, the mobile device does not wait until the updated student weights arrive; it just proceeds to the next non-key frame *asynchronously* with the slightly outdated student, thereby exploiting temporal coherence further. Finally, the stride to the next key frame is determined by a simple but adaptive algorithm based on the student's performance after distillation.

Sending the updated weights to the mobile device incurs little overhead because the student is very small by design. For instance, in our experiment, the student model is even smaller than a single video frame. Still, we propose *partial distillation*, a technique that further reduces the transmitted data size, and apply it to ShadowTutor. That is, upon distilling knowledge from the teacher, only a subset of the student's parameters is trained. This technique accelerates knowledge distillation and allows only the updated part of the student to be transmitted across the network. Moreover, surprisingly, partial distillation yields generally higher accuracy and further reduces the number of network communications compared with adapting all parameters.

We evaluate the effectiveness of ShadowTutor on HD video semantic segmentation, a challenging video computer vision task that requires dense predictions of pixel-level class probabilities. Evaluations show that compared with naive DNN offloading, ShadowTutor reduces network data transfer by 95% on average and improves throughput by over 3x. Also, due to asynchronous inference, ShadowTutor effectively retains throughput under low available network bandwidth.

In sum, we make the following contributions:

- We design ShadowTutor, a distributed video DNN inference framework that reduces the number of network transmissions and the data size of each.
- We allow ShadowTutor to be robust to changes in network conditions by adopting asynchronous inference.
- We aid the configuration of ShadowTutor by providing analytic models of its network traffic and throughput in terms of system parameters and component measurements.
- We adapt ShadowTutor to HD video semantic segmentation, a real-life application, and evaluate its performance in terms of throughput, network traffic, accuracy, and robustness.

The rest of this paper is organized as follows. We review previous approaches in section 2, and provide background regarding our work in section 3. We explain the core idea and mechanism of ShadowTutor in section 4, and show how it is adapted specifically to video semantic segmentation in section 5. Then, we evaluate our framework in section 6. Finally, we introduce related work in section 7 and conclude the paper in section 8.

2 PREVIOUS APPROACHES

Due to the resource constraints of mobile devices, DNN computations are often offloaded to central cloud servers. Neurosurgeon [17] partitions a given DNN and collaboratively executes it between the cloud and the server. It searches for an optimal partition point in terms of latency and power consumption. Eshratifar et al [8] also seeks to partition DNNs, but by deriving optimal conditions for client power consumption and cloud resource constraints. MCDNN [12] generates modified and specialized versions of the original model to trade-off accuracy for improved resource usage. However, it only deals with models that perform image classification, requires the execution of the model for every image in the training set for specialization (pre-forwarding), and the generated model is mostly limited to being a layer-wise modification of the original model. Moreover, adopting these approaches still require the communication of every frame across the network, thereby pressuring network traffic greatly.

Another work by Yang et al. [31] splits video frames into partitions that are distributed to slaves. It utilizes optical flow, i.e. the movement vectors between two frames, to exploit the temporal coherence between frames. However, their system still requires the communication of every single frame between the master and the slaves, since they utilize temporal coherence only to occasionally reduce the amount of computation, rather than reducing network traffic.

3 BACKGROUND

3.1 Knowledge Distillation

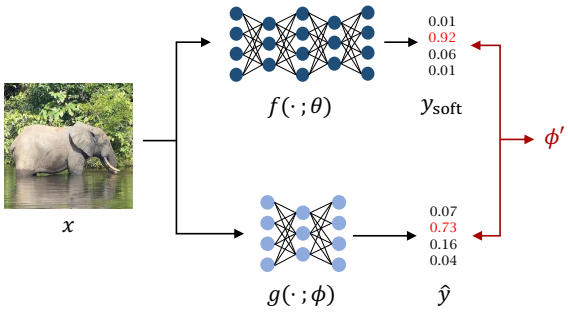


Figure 1: Inference time knowledge distillation. The student's parameters are updated by viewing the teacher's ouptut as the label.

Knowledge distillation [15] was initially proposed as a means of model compression. Viewing a large teacher model's output as a *soft target*, it formulated a knowledge distillation loss between the output of a small student model and the soft target. Compared with *hard targets*, which are one-hot vectors from the dataset, soft targets provide much more information about the input data, and allow less variance in the gradients of the student model.

Figure 1 illustrates the concept of knowledge distillation applied to image classification. Since it assumes inference time, the actual label y_{hard} is not available. Thus, the student learns just from the output of the teacher model y_{soft}. Is is also the case for ShadowTutor.

3.2 Semantic Segmentation

Semantic segmentation is in other words *pixel-wise classification*. There is a fixed set of classes, including the background class. Then, the model is required to classify each pixel of its input image to one of the given classes. Thus, the output of semantic segmentation has the same spatial size as the input image.

The result of semantic segmentation is often evaluated with mean IoU (Intersection over Union). With $pred_c$ the set of pixels classified as class c and $label_c$ the set of pixels with ground truth class c, the IoU for class c is defined as

$$\text{IoU}_c = \frac{|pred_c \cap label_c|}{|pred_c \cup label_c|}. \quad (1)$$

The IoU is computed for each class in the ground truth label and averaged to yield the mean IoU performance metric. Mean IoU thus lies between 0 and 1, with 1 being the best possible score.

4 SHADOWTUTOR

Figure 2 shows an overview of ShadowTutor. All frames are processed sequentially in temporal order. Determined by the key frame scheduling policy, if the current frame is a non-key frame, its inference is handled entirely on-device by a lightweight student model. If the current frame is a key frame, it is sent to the server. Using the key frame, the server updates the a copy of the student's weights via knowledge distillation from the teacher, and returns the updated weights to the client. Network communication occurs only at sparse key frames, allowing ShadowTutor to reduce the amount of network traffic drastically.

4.1 System Components

Stripping away the distributed nature from ShadowTutor, there are five essential components that comprise the system: video data, teacher inference, student inference, student training, and key frame striding.

4.1.1 Video data. Video data reside in the mobile device, and the majority of the frames do not leave its home. The video can either be in an internal storage or fetched from the camera in real-time. Since each frame is traversed by the system in strict temporal order without look-back, frames can be readily discarded once they are processed.

4.1.2 Teacher inference. By *teacher*, we refer to a heavy but high-quality neural network deployed to the server. Since ShadowTutor is an inference system, labels for the input data do not exist. Thus, it is important to employ a sufficiently good teacher model that can generalize well to unseen video scenes. Also, the teacher should be pre-trained on a dataset relevant to the task. Note that teacher inference is done only for the key frames transmitted to the server.

4.1.3 Student inference. By *student*, we refer to a lightweight neural network designed to run on the mobile device. The student model need not be deployed to the mobile device when ShadowTutor is installed, because the server can simply supply the student weights when the system starts. The student should also be pre-trained on relevant data, most likely the data used to pre-train the teacher. Pre-training can be expensive, but it is a one-time cost.

Algorithm 1: Student Training

Input: Student model *student*, key frame *frame*, pseudo-label *label*
Output: Trained student model and corresponding metric

Function `Train`(*student*, *frame*, *label*):

```
prediction ← Forward(student, frame)
best_metric ← ComputeMetric(prediction, label)
best_student ← student
if best_metric < THRESHOLD then
    for i ← 1 to MAX_UPDATES do
        loss ← ComputeLoss(prediction, label)
        gradients ← PartialBackward(loss)
        student ← OptimStep(student, gradients)
        prediction ← Forward(student, frame)
        metric ← ComputeMetric(prediction, label)
        if metric > best_metric then
            best_metric ← metric
            best_student ← student
        end
        if metric > THRESHOLD then
            break
        end
    end
end
return (best_student, best_metric)
```

For non-key frames, student inference is the only operation done on them. Hence, student inference latency determines the steady state throughput of ShadowTutor.

4.1.4 Student training. With the teacher's inference result, deemed the pseudo-label, the student model is trained. Since the student model is very small, it lacks generalization power to perform well on any video stream it encounters. However, in our setting, there is no need for generalization; we only need to do well for the video at hand. Thus, by training the student with sparse key frames, we overfit the student model to the current video stream.

Algorithm 1 shows the process of student training. Optimization steps (most simply gradient descent) for the student are taken until either the performance metric (e.g. accuracy, mean IoU) exceeds `THRESHOLD`, or the number of training steps reach `MAX_UPDATES`. The best performing student and the corresponding metric is returned. `THRESHOLD` and `MAX_UPDATES` are algorithmic parameters. The former denotes an acceptable level of student performance, and the latter the maximum number of optimization steps. Especially, the higher the `THRESHOLD`, the more the expected number of optimization steps, since it becomes harder for the student to break out of the training loop. Thus, increasing either of the two potentially increases student performance, but the system's throughput decreases. Moreover, we perform *partial distillation*, as denoted by the function `PartialBackward`. We discuss this in greater detail in section 4.2.

We emphasize that this process be distinguished from student pre-training; student training is done repetitively during system

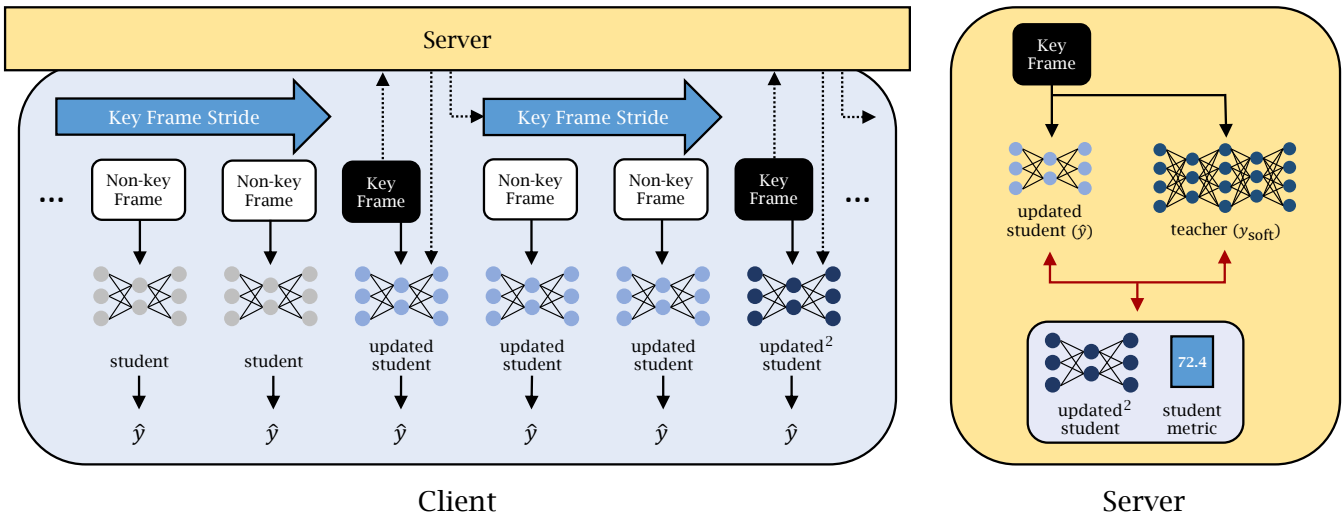


Figure 2: High-level view of ShadowTutor. Dotted arrows denote network communication, while other arrows denote data transfers within a device. Each frame is processed one by one sequentially. Non-key frame inferences are done on-device with a small student network. Sparse key frames are used to update the student's weights via knowledge distillation from a teacher.

Algorithm 2: Compute Next Stride

Input: Current stride *stride*, student performance metric *metric* $\in [0, 1]$
Output: Next stride

Function `NextStride`(*stride*, *metric*):

```
1  if metric < THRESHOLD then
2      ratio ← metric/THRESHOLD
3  else
4      ratio ←
         (metric − 2∗THRESHOLD+1)/(1−THRESHOLD)
5  end
6  stride ← ratio ∗ stride
7  stride ← Clamp(stride, MIN_STRIDE, MAX_STRIDE)
8  return stride
```

runtime, and student pre-training is done exactly once when the system is first organized.

4.1.5 Key frame striding. After training the student on the current key frame, the distance to the next key frame must be determined. It is logical to change the stride, which defines how often to run student training, based on the student's current performance. However, existing literature regarding key frame striding provides no suitable solution since they are either not adaptive or simplistic (fixed stride [32], exponential back-off [20]), or overly complex (clockwork networks [26], LSTM [2]) for mobile devices. Thus, we aim design a key frame scheduling algorithm that is sufficiently simple but still adaptive, which is presented in 2. Here, the *ratio* of the next stride to the current one is first determined. Line 2 is a linear function of *metric* that connects points $(0, 0)$ and $(\texttt{THRESHOLD}, 1)$, and line 4 another linear function connecting $(\texttt{THRESHOLD}, 1)$ and $(1, 2)$. Hence, if the student performs better than `THRESHOLD`, the distance to the next key frame is elongated, and otherwise shortened. Finally, to prevent the stride from vanishing or diverging, we clamp the stride with `MIN_STRIDE` and `MAX_STRIDE`. These two parameters must be selected based on the data at hand. For example, if the video has very low FPS and changes quickly even in a few frames, it would be better to choose small numbers for them. Also, too small values will lead to increased network traffic and deteriorate throughput, while too large values will lead to low student performance. We analytically model network traffic and throughput in terms of these algorithm parameters in section 4.4.

4.2 Partial Knowledge Distillation

Instead of changing every parameter of the student, which we call *full* distillation, we suggest to freeze the *front part* of the network, and only train the rest. This technique is beneficial in terms of latency, memory consumption, and network traffic. Moreover, it is potentially so for student performance (accuracy).

The reduction in latency and memory consumption is immediate. Consider the process of back propagation. Gradients with respect to trainable (not frozen) parameters are computed via chain rule from the loss to the leaf of the computation graph. Thus, if the front part of a neural network is entirely frozen, gradient computation can stop in the middle of the network. This leads to decreases in the amount of computation and latency. Also, memory need not be allocated for the gradients of frozen parameters.

The reduction in network traffic is also evident. Since both the server and the client agree that only a subset of the student's weights will be adapted, it suffices to communicate only the weights that changed. Sending the student weights already incurs little overhead because the student is small by design. Still, partial distillation reduces this overhead further.

Algorithm 3: ShadowTutor-Server

Input: Student model *student*, teacher model *teacher*

```
1 ToClient(student)
2 while forever do
3 |   FromClient(frame)
4 |   label ← Forward(teacher, frame)
5 |   student, metric ← Train(student, frame, label)
6 |   ToClient(UpdatedPart(student), metric)
7 end
```

Algorithm 4: ShadowTutor-Client

Input: Target video stream *video*
Output: Student predictions

```
1  stride ← MIN_STRIDE
2  step ← stride          // First frame as key frame
3  updated ← true   // Whether student recv is done
4  FromServer(student)
5  foreach frame in video do
6  |   if step = stride then                  // Key frame
7  |   |   ToServerAsync(frame)
8  |   |   async_recv ←
   |   |    FromServerAsync(student_diff, metric)
9  |   |   step ← 0
10 |   |   updated ← false
11 |   end
12 |   prediction ← Forward(student, frame)
13 |   step ← step + 1
14 |   if not updated then
15 |   |   if step = MIN_STRIDE then
16 |   |   |   WaitUntilComplete(async_recv)
17 |   |   end
18 |   |   if Completed(async_recv) then
19 |   |   |   student ← ApplyUpdate(student, student_diff)
20 |   |   |   stride ← NextStride(stride, metric)
21 |   |   |   updated ← true
22 |   |   end
23 |   end
24 end
```

The claim of better student performance is rather subtle. Most neural networks are consisted of a front-end feature extractor, which transforms the input data to a more useful representation, and the back-end network, which maps the representation to a form suitable for the downstream task. Thus, given only a limited number of training steps, jointly training both the feature extractor and the back-end network will be costly and unstable. This is because the front-end network will emit different features every distillation step, and the back-end network must constantly adapt to the change in order to correctly map the feature to the task representation. Thus, adapting only the weights of the back-end network with a fixed feature distribution can be more stable. In a sense, with only a small budget for exploration, it is better to invest on exploitation. Further, drawing reason from existing literature, it is a common practice to freeze the front part of a pre-trained network even for transfer learning [28]. Transfer learning is far more challenging compared with our setting, because it aims to adapt a pre-trained network towards a different task, whereas we stay on the same task.

We empirically validate these claims in section 6 through a series of comparisons with full distillation.

4.3 ShadowTutor

Now, we aggregate the system components into ShadowTutor. Algorithm 3 and 4 shows the runtime operations of the server and the mobile device (client) respectively. ShadowTutor offloads teacher inference and student training to the server. Needless to say, teacher inference is an unacceptable workload for mobile devices, so offloading is a natural choice. Student training on mobiles devices is feasible, but it still incurs significant delay. Also, student training blocks student inference; the mobile device cannot proceed with non-key frame inference during student training. This causes large fluctuations in the system's throughput. On the other hand, for the server, training the student only adds marginal overhead since the student only has a small number of parameters.

ShadowTutor reduces network traffic considerably. This is because network communications between the server and the client only occurs at key frames, as opposed to naive offloading, which requires communication for every frame. In addition, thanks to partial distillation, the data size of each network communication is cut down further. Thus, ShadowTutor reduces network traffic in terms of the number of network transmissions and the number of packets in each transmission.

Importantly, offloading student training reveals an opportunity for *asynchronous* inference. As shown in line 7 and 8 of algorithm 4, the mobile device sends key frames and receives the updated student parameters in a non-blocking fashion. That is, the mobile device sends the key frame to the server, and without waiting for the updated parameters, proceeds on to the next (non-key) frame. This approach further exploits the temporal coherence between frames by assuming that even after a key frame, the student weights are still usable. The updated parameters are awaited for a maximum `MIN_STRIDE` steps, since the next key frame stride may become `MIN_STRIDE`. This is the key mechanism that allows ShadowTutor to be robust to adverse network conditions. That is, ShadowTutor can mitigate delays in network transfer by just proceeding further into future frames.

4.4 Network Traffic and Throughput Bounds

In this section, we model ShadowTutor's network traffic (amount of data transferred per unit time) and throughput (number of frames processed per unit time) and derive formulae for their lower and upper bounds. We use the notations defined in table 1. Note that t_{net} and s_{net} regard the transmission of one key frame and the corresponding updated student parameters.

Both network traffic and throughput rely on the system's total execution time, which we first model. ShadowTutor is designed

Table 1: Notations used for ShadowTutor. Those in the first block are identified after system execution, and the second based on system component decisions.

Symbol	Definition
n	**n**umber of frames processed
d	number of **d**istillation steps taken
k	number of **k**ey frames
t_{si}	latency of **s**tudent **i**nference
t_{sd}	latency of one **s**tudent **d**istillation step
t_{ti}	latency of **t**eacher **i**nference
t_{net}	**net**work latency associated with one key frame
s_{net}	**net**worked data **s**ize associated with one key frame

to perform asynchronous inference for at most MIN_STRIDE many frames after a key frame. However, a mobile device may either be able to execute student inference and network operations entirely in parallel, or it may not support any form of concurrency. Therefore, t_c, the execution time of MIN_STRIDE frames after a key frame, is within the following bounds:

$$\begin{aligned} t_c &\geq \max(\texttt{MIN\_STRIDE} \times t_{si}, t_{net} + t_{ti}) \\ t_c &\leq \texttt{MIN\_STRIDE} \times t_{si} + t_{net} + t_{ti} \end{aligned} \tag{2}$$

Then, with t_c, the total execution time for processing n frames can be modelled as follows:

$$t_{tot} = (n - k \times \texttt{MIN\_STRIDE})t_{si} + dt_{sd} + kt_c. \tag{3}$$

Now, we obtain a general formula for network traffic by dividing the total size of data transfer by the total execution time:

$$\frac{ks_{net}}{t_{tot}} = \frac{ks_{net}}{(n - k \times \texttt{MIN\_STRIDE})t_{si} + dt_{sd} + kt_c}. \tag{4}$$

Minimum network traffic is achieved when key frames are least frequent, the execution time for each key frame is longest, and the client completely lacks concurrency. That is,

$$k = \frac{n}{\texttt{MAX\_STRIDE}}, \tag{5}$$

$$d = k \times \texttt{MAX\_UPDATES}, \tag{6}$$

and

$$t_c = \texttt{MIN\_STRIDE} \times t_{si} + t_{net} + t_{ti} \tag{7}$$

hold. Thus, from equation 4, the network traffic lower bound is:

$$\frac{s_{net}}{\texttt{MAX\_STRIDE} \times t_{si} + \texttt{MAX\_UPDATES} \times t_{sd} + t_{ti} + t_{net}}. \tag{8}$$

On the other hand, network traffic is maximum when key frames are as frequent as possible, the execution time for each key frame is shortest, and the client is capable of handling student inference and network operations entirely in parallel. That is,

$$k = \frac{n}{\texttt{MIN\_STRIDE}}, \tag{9}$$

$$d = 0, \tag{10}$$

and

$$t_c = \max(\texttt{MIN\_STRIDE} \times t_{si}, t_{net} + t_{ti}) \tag{11}$$

hold. Especially, equation 10 holds because distillation can be entirely skipped based on the student's initial metric (see line 4 in algorithm 1). Again, from equation 4, the network traffic upper bound is:

$$\frac{s_{net}}{\max(\texttt{MIN\_STRIDE} \times t_{si}, t_{net} + t_{ti})}. \tag{12}$$

We can also obtain a general formula for throughput by dividing the number of processed frames by the execution time:

$$\frac{n}{t_{tot}} = \frac{n}{(n - k \times \texttt{MIN\_STRIDE})t_{si} + dt_{sd} + kt_c}. \tag{13}$$

The throughput lower bound is achieved when the total execution time is the longest. In such case, equations 9, 6, and 7 hold. Thus, from equation 13, the throughput lower bound is

$$\frac{\texttt{MIN\_STRIDE}}{\texttt{MIN\_STRIDE} \times t_{si} + \texttt{MAX\_UPDATES} \times t_{sd} + t_{ti} + t_{net}}. \tag{14}$$

On the other hand, the throughput upper bound is achieved when the total execution time is the shortest. In that case, equations 5, 10, and 11 hold. Again, from equation 13, the throughput upper bound is

$$\frac{\texttt{MAX\_STRIDE}}{(\texttt{MAX\_STRIDE} - \texttt{MIN\_STRIDE})t_{si} + \max(\texttt{MIN\_STRIDE}t_{si}, t_{net} + t_{ti})}. \tag{15}$$

Notice that in all lower and upper bound formulae, only algorithm parameters, latency measurements, and data size remain. Thus, ShadowTutor allows the estimation of the system's network bandwidth requirement and throughput prior to actually implementing and running the entire system. We use these bounds to determine algorithm parameters in section 5.3.

5 SHADOWTUTOR FOR VIDEO SEMANTIC SEGMENTATION

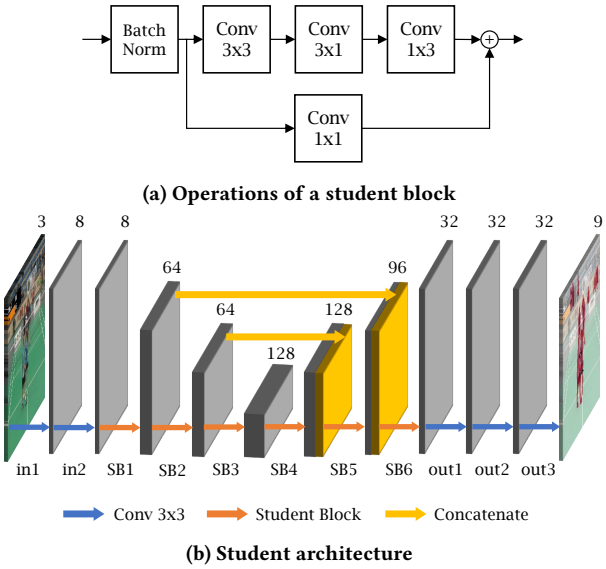
5.1 Experiment setup

As the server, we use a desktop computer equipped with an AMD Ryzen 7 3700X CPU, one NVIDIA RTX 2080ti GPU, and 32 GB memory. As the client, we use the NVIDIA Jetson Nano embedded board [21], equipped with a quad-core ARM A57 CPU, a 128-core Maxwell GPU, and 4 GB memory. Jetson Nano can deliver up to 472 GFLOPS for 32-bit floating points, which is not an unrealistic number for modern mobile devices. For example, Google Pixel 4's Qualcomm Snapdragon 855 can deliver up to 954.7 GFLOPS (32-bit) with its built-in Adreno 640 GPU [9]. As to network configurations, we limit both uplink and downlink bandwidth to 80 Mbps, assuming strong Wi-Fi connection. We implement the system with OpenMPI [10], PyTorch [23], and Detectron2 [30].

5.2 System Component Decisions

We target videos with 25–30 FPS from the Long Video Segmentation (LVS) dataset [20] [1]. We use high resolution (720p HD) videos in order to pressure the overall load of the system. The LVS dataset is labeled with 8 actively moving object classes (person, bicycle, automobile, bird, dog, horse, elephant, and giraffe), making accurate segmentation challenging because no object class remains stationary in the scene. The movement of the camera is either fixed, moving, or egocentric (shot from a camera attached to a person's head or chest). Also, the main scenery of each video is one of the three: animals, people, or street.

[1] Obtained from https://olimar.stanford.edu/hdd/lvsdataset/.



(a) Operations of a student block

(b) Student architecture

Figure 3: The student model is a small fully-convolutional network.

We choose Mask R-CNN [13] as the teacher neural network, because the LVS dataset was actually labelled by selecting videos that Mask R-CNN performs particularly well on. The teacher's weights (pre-trained on the Microsoft COCO dataset [19]) are adopted from the Detectron2 framework.

The student neural network is a simple fully-convolutional network. Figure 3a shows the operations of a student block, and 3b the layer compositions of the student. The low resolution feature maps from SB2 and SB1 are concatenated to the input of SB5 and SB6, respectively. We pre-trained the student on COCO for 30 epochs with the Adam [18] optimizer. As to partial distillation, we freeze the student from the first layer to SB4, only computing gradients until SB5. This way, the trained parameters amount to 21.4% of all parameters.

Counting the raw number of parameters, the teacher has 44.34 million parameters, whereas the student has 0.48 million. Thus, the teacher is 100x larger than the student. However, as we will see in section 6.3, the student can approach the teacher's performance through shadow education.

Model-level optimizations such as using more efficient operations (as in MobileNet [25]) or performing quantization or pruning on weights [11] can be applied to the student. Further, teacher inference can be accelerated with inference optimizations such as TensorRT [22]. However, although these techniques may improve throughput and decrease network traffic, we exclude them in our study. This is because the effect of such techniques differ widely based on implementation, platform, and hardware.

Finally, we decide upon the actual process of knowledge distillation. As pointed out by the LVS dataset paper, videos in the dataset have an excess amount of background class pixels. Due to this class imbalance, training the student with vanilla cross entropy may bias the student towards predicting every pixel as background. Thus, we directly adopt their loss weighting approach, which scales the cross-entropy loss of pixels near and within non-background objects by a factor of 5. Partial knowledge distillation to the student parameters is done with the Adam optimizer with a learning rate of 0.01.

5.3 Algorithm Parameter Decisions

We first determine `THRESHOLD` using the performance of existing video semantic segmentation models. Since the LVS dataset has no official leaderboard available, we consider a similar dataset called Cityscapes [7]. Since the state-of-the-art approach for Cityscapes records an mIOU of 0.845 at the time of research, we set `THRESHOLD` to 0.8.

As to `MIN_STRIDE` and `MAX_STRIDE`, we consider the FPS of the videos. We have selected videos with an FPS of 25–30. Following this, we set `MIN_STRIDE` to 8 and `MAX_STRIDE` to 64. That is, within 8 frames, we postulate that the objects in the scene will have hardly changed, and re-training within 8 frames would be meaningless. On the other hand, after 64 frames or 2.5 seconds in a video clip, we assume that the objects will have moved significantly, and the student at least needs to be tested.

Finally, we determine `MAX_UPDATES` using the throughput bounds derived in equation 14 and 15. Using the notations defined in table 1, our experiment setting gives $t_{si} = 0.143$, $t_{sd} = 0.013$, $t_{ti} = 0.044$, and $t_{net} = 0.303$ in seconds. Then, equation 15 yields a maximum throughput of 6.99 FPS. In order to keep the difference between the theoretical maximum and minimum throughput within 2 FPS, we find the largest `MAX_UPDATES` value that gives a throughput lower bound larger than 5 FPS, which is 8.

6 EVALUATION

Using the configurations from section 5, we show ShadowTutor's effectiveness. Specifically, we investigate the advantages of ShadowTutor in terms of throughput, network traffic, and accuracy in sections 6.1, 6.2, and 6.3, respectively. Then, we push ShadowTutor to the limits in terms of network bandwidth in section 6.4 and temporal coherence in section 6.5.

ShadowTutor is mainly compared with naive offloading (sending every frame to the server and retrieving inference results). Also, we compare partial distillation and full distillation to justify the design of ShadowTutor. All experiments are performed on the first 5000 frames of each video stream (about 200 seconds). Every ShadowTutor experiment, whether partial or full distillation, begin from the same pre-trained student checkpoint.

In or experiments, we only employ one type of teacher model: the Mask R-CNN. This is because the student, who learns from the teacher, is only interested in the final output of the teacher, regardless of all the intermediate operations.

6.1 Throughput

Table 2 summarizes the latency of one distillation step (ms) and the mean number of distillation steps for partial and full distillation. Partial distillation reduces both the latency and the number of distills, contributing to the throughput of the entire system. This result empirically supports the claim that since the number of

Table 2: Execution time and mean number of distillation steps

Distillation	Partial	Full
One step (ms)	13	18
Mean # of steps	3.83	4.44

Table 3: Frames processed per second (FPS) and execution time (s) in parenthesis

Camera	Scene	Partial	Full	Naive
fixed	animals	6.55(762.5)	6.21(804.5)	2.09(2391.3)
fixed	people	**6.60(757.4)**	6.43(777.0)	2.09(2391.3)
fixed	street	6.50(768.8)	5.95(840.5)	2.09(2391.3)
moving	animals	6.57(760.5)	6.27(796.5)	2.09(2391.3)
moving	people	**6.59(758.5)**	6.36(785.8)	2.09(2391.3)
moving	street	6.41(780.2)	5.55(901.0)	2.09(2391.3)
egocentric	people	6.57(760.5)	5.89(848.5)	2.09(2391.3)
average		**6.54(764.1)**	6.08(822.0)	2.09(2391.3)

Table 4: Data transmitted on each key frame (MB).

Direction	Partial	Full	Naive
To Server	2.637	2.637	2.637
To Client	**0.395**	1.846	0.879
Total	**3.032**	4.483	3.516

Table 5: Key frames ratio (%) and network traffic (Mbps)

Camera	Scene	Key frame ratio			Network traffic	
		Partial	Full	Naive	Partial	Naive
fixed	animals	4.73	4.60	100.0	7.51	58.51
fixed	people	**1.96**	2.42	100.0	**3.14**	58.51
fixed	street	7.78	7.43	100.0	12.27	58.51
moving	animals	2.55	2.29	100.0	4.06	58.51
moving	people	**3.45**	4.12	100.0	5.51	58.51
moving	street	11.70	11.48	100.0	18.19	58.51
egocentric	people	**5.46**	9.75	100.0	8.70	58.51
average		**5.38**	6.01	100.0	**6.19**	58.51

Table 6: Mean IoU of various settings. Wild = pre-trained student on its own, P = partial distillation, F = full distillation, digit (1 or 8) = number of delayed frames before receiving updated student weights.

Camera	Scene	Wild	P-1	P-8	F-1	Naive
fixed	animals	14.34	74.31	73.27	74.47	100.0
fixed	people	13.91	81.69	81.39	81.36	100.0
fixed	street	17.28	**70.26**	69.01	63.60	100.0
moving	animals	22.31	74.94	73.80	75.21	100.0
moving	people	17.62	74.82	74.06	75.55	100.0
moving	street	18.65	**60.48**	58.61	52.94	100.0
egocentric	people	14.80	**70.42**	68.87	61.41	100.0
average		16.99	**72.42**	71.29	69.22	100.0

training steps is limited, it is quicker to exploit a fixed distribution of features than to explore for better ones.

Table 3 lists the actual throughput of the system (frames processed per second) and the total execution time. As expected, partial distillation outperforms full distillation in every category. Moreover, ShadowTutor shows an improvement greater than 3x over naive offloading. This is especially because ShadowTutor only communicates with the server on key frames, and thus drastically reduces the latency for networking.

6.2 Network Traffic

We investigate the reduction of network traffic in terms of the amount of data transfer per key frame and the ratio of key frames to all frames.

Table 4 shows the amount of data transfer (in MB) per key frame. Since it suffices to send only the updated part of the student, partial distillation reduces network traffic compared with full distillation. Against naive offloading, which sends the teacher prediction to the client, ShadowTutor reduces the amount of data transfer by 13.77% per key frame, because the size of the student is even smaller than one video frame.

Table 5 summarizes the proportion of key frames and the actual network traffic in Mbps. The smaller the key frame proportion, the less frequent the network communication. Partial distillation generally performs better than full distillation, and strictly better than naive offloading. Especially, for the fixed-people category, the number of network communications is only 1.96% compared with the naive offloading scheme, yielding a surprising 98% reduction.

The effects of the two reductions are multiplicative. Thus, compared with naive offloading, ShadowTutor reduces the amount of network transfer per frame by 98.3% at most, and 95.3% on average.

On the other hand, the reduction in network traffic (amount of networked data per unit time) is coupled with the improvement in throughput, showing a reduction of 89.4% on average. We especially note that network traffic has improved even if throughput had a threefold improvement.

Finally, with the current configuration, the network traffic bounds computed with equations 8 and 12 are 2.53 Mbps and 21.2 Mbps, respectively. From table 5, all network traffic values obey the bounds, and the bounds are quite tight, proving their usefulness.

6.3 Accuracy

Table 6 shows the mean Intersection over Union (mIoU) of various experiment settings. The mIoU of *every* frame (key and non-key frames) is averaged to show that the student can leverage temporal coherence to accurately perform inference on non-key frames. Note that all accuracy values are evaluated against the teacher (Mask R-CNN) output, which is why the naive approach always achieves perfect accuracy. However, we emphasize that the LVS dataset has

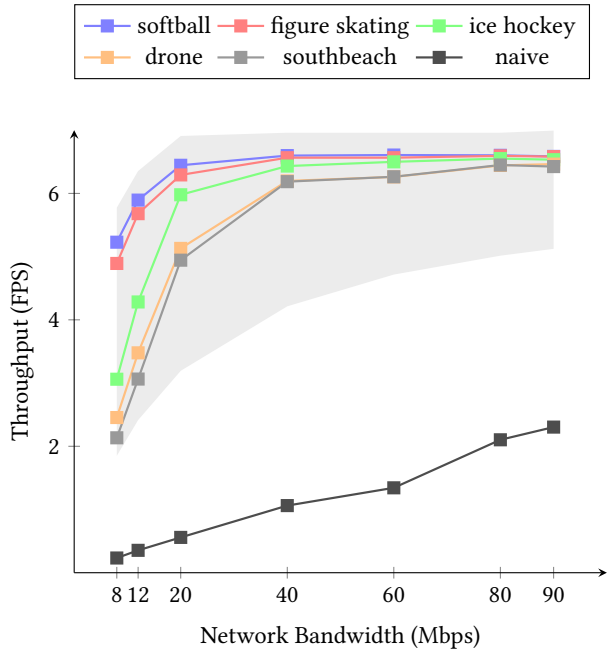


Figure 4: Network bandwidth and system throughput

been labelled with the Mask R-CNN. Thus, in our case, we are measuring the accuracy against the label in effect.

First, to show the need for shadow education, we run the pre-trained student on every frame without any supervision from the teacher (denoted as Wild). As expected, its accuracy suffers greatly, approaching the accuracy of random guessing. This is because the student is too small to generalize to all kinds of scenes.

Next, we show the effect of shadow education. Recall that the mobile device receives the updated weights in a non-blocking fashion, mitigating the effect of delays in network transfer. Thus, we measure accuracy when there is the least delay (1 frame, P-1) and the most (8 frames, P-8). ShadowTutor approaches the accuracy of the teacher with a student 100x smaller, proving the effectiveness of knowledge distillation. Moreover, asynchronous inference hardly hurts accuracy, showing that slightly outdated weights are still useful due to temporal coherence.

Lastly, we compare partial distillation (P-1) and full distillation (F-1). Overall, partial distillation is more accurate. When partial distillation is better, it outperforms full distillation significantly, thereby providing an overall stable level of accuracy.

6.4 Robustness to Network Conditions

Fluctuations often happen during network communications between the cloud data center and the client. Thus, in this experiment, we investigate the effect of reduced available network bandwidth. Specifically, we set the bandwidth of the system to 90, 80, 60, 40, 20, 12, and 8 Mbps, and examine the throughput of the system.

Figure 4 shows the change in throughput against network bandwidth for ShadowTutor and naive offloading. For ShadowTutor, we selected five video streams with different key frame proportions;

Table 7: Mean IoU and key frame ratio for 7 FPS videos. Digit (1 or 8) = number of frame delays before receiving updated student weights. Key frame proportion is in %.

Camera	Scene	Partial-1	Partial-8	Key frame
fixed	animals	62.72	61.86	6.59
fixed	people	80.44	80.08	1.97
fixed	street	63.78	62.51	8.9
moving	animals	68.63	66.78	4.84
moving	people	73.66	72.91	4.15
moving	street	48.92	46.99	12.34
egocentric	people	67.57	66.09	5.44
average		66.53	65.31	6.32

softball has the least key frames (1.72%), and southbeach (street CCTV) has the most (12.4%).

The throughput of naive offloading decreases immediately in the face of low network bandwidth because it has no mechanism to mitigate the increase in network latency. On the contrary, the throughput of ShadowTutor remains remarkably stable until 40 Mbps, which is half of the original bandwidth. For videos that have a small proportion of key frames, throughput is retained even until 20 Mbps, since network latency takes up only a small fraction among all latency components. Videos with more key frames lose throughput more quickly, but only by 3x even if the network bandwidth is 10x narrower.

The region colored in gray represent the throughput bounds computed with equations 14 and 15. All throughput values obey the bounds. Especially, for low bandwidth settings, network latency dominates among all latency components, reducing the variation in throughput brought about by the degree of concurrency supported by the mobile device.

ShadowTutor's robustness to the reduction in network bandwidth comes from asynchronous inference. In effect, as long as the network latency is shorter than the inference latency of `MIN_STRIDE` many frames, ShadowTutor can hide the network latency almost completely. However, when the network latency is longer, the reduction in network bandwidth begins to take a more direct impact to the system's throughput since asynchronous inference can no longer serve as a buffer.

6.5 Feasibility of Real-Time Inference

Finding promise from 25–30 FPS videos, we test ShadowTutor with videos with less temporal coherence. Specifically, for every video, we re-sample the frames such that all videos have an FPS of 7. Thus, by matching the input video's frame rate with ShadowTutor's throughput, we simulate the *real-time inference* of frames fetched from the mobile device's camera.

Table 7 shows the mean IoU and key frame proportion of the first 5000 frames of the re-sampled videos. Surprisingly, even if the time distance between adjacent frames is elongated by four times, ShadowTutor yields an average accuracy drop of less than 6%p and key frame proportion increase of less than 1%p. Therefore, this shows the feasibility of applying ShadowTutor to real-time inference applications. With optimized students, e.g. those that utilize

weight quantization/pruning or employ more efficient operations, ShadowTutor will be able to handle higher frame rate videos in real-time, and its accuracy will increase further due to stronger temporal coherence.

7 RELATED WORK

We briefly survey works related to extending or replacing parts of ShadowTutor.

The concept of knowledge distillation has been applied to computer vision tasks widely. Chen et al. [5] applied knowledge distillation to creating efficient object detection models for images. Mullapudi et al. [20] extended knowledge distillation to videos. Bajestani et al. [2] extends the previous work by designing an LSTM-based key frame selection method and a teacher-bounded loss function. These works do not consider the context of mobile or distributed computing. Still, they provide insight into knowledge distillation and the possibility of direct application to ShadowTutor.

We may delve further into the possibility of extending ShadowTutor's knowledge distillation process. The original knowledge distillation paper [15] also proposed to distill knowledge from an *ensemble* of different teacher models. Moreover, data distillation [24] proposes to use only a single teacher, but to ensemble its outputs on the same image but with different transformations applied. Chung et al. [6] proposes to apply knowledge distillation at the *feature*-level through adversarial training.

8 CONCLUSION AND FUTURE WORK

In this work, we developed ShadowTutor, a distributed video DNN inference framework that encodes the temporal coherence in the video at hand into the parameters of a small student model through intermittent partial knowledge distillation. Evaluations show its advantages in terms of network traffic, throughput, accuracy, and robustness. Moreover, ShadowTutor achieves this through a simple and elegant split between the server and the client, and without complex DNN partitioning, server-side scheduling, or model-level optimizations tailored to the experiment devices.

While our work focused on developing a distributed framework for video data, ShadowTutor can also be extended to tasks other than video computer vision. That is, there are plenty of *sequence data*, i.e. a collection of *data points* that are temporally coherent, that are handled with DNNs. Examples of such sequence data include speech signals from a single speaker, a sentence that requires translation, or a series of item recommendation requests from a single user. Thus, by exploiting the temporal coherence embedded in various types of sequence data through intermittent knowledge distillation, ShadowTutor has the potential to be extended to any type of sequence data. Upgrading the knowledge distillation process, designing appropriate model architectures for the teacher and the student, and resolving the issues that may arise in the process will serve as a good future research direction.

ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2020R1A2B5B02001845).

REFERENCES

[1] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha. 2017. Real-Time Video Analytics: The Killer App for Edge Computing. *Computer* 50, 10 (2017), 58–67.

[2] M. F. Bajestani and Y. Yang. 2020. TKD: Temporal Knowledge Distillation for Active Perception. In *The IEEE Winter Conference on Applications of Computer Vision*. 953–962.

[3] V. Bazarevsky and A. Tkachenka. 2018. *Mobile Real-Time Semantic Segmentation*. Retrieved March 13, 2020 from https://ai.googleblog.com/2018/03/mobile-real-time-video-segmentation.html

[4] W. Chan, N. Jaitly, Q. Le, and O. Vinyals. 2016. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 4960–4964.

[5] G. Chen, W. Choi, X. Yu, T. Han, and M. Chandraker. 2017. Learning efficient object detection models with knowledge distillation. In *Advances in Neural Information Processing Systems*. 742–751.

[6] I. Chung, S. Park, J. Kim, and N. Kwak. 2020. Feature-map-level Online Adversarial Knowledge Distillation. *arXiv preprint arXiv:2002.01775* (2020).

[7] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. 2016. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[8] A. Eshratifar and M. Pedram. 2018. Energy and performance efficient computation offloading for deep neural networks in a mobile cloud computing environment. In *Proceedings of the 2018 on Great Lakes Symposium on VLSI*. 111–116.

[9] Andrew G. 2019. *Mobile GPU approaches to power efficiency*. Retrieved March 3, 2020 from https://www.highperformancegraphics.org/wp-content/uploads/2019/hot3d/mobile_gpu_power_and_performance.pdf

[10] E. Gabriel, G. Fagg, G. Bosilca, T. Angskun, J. Dongarra, J. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. Castain, D. Daniel, R. Graham, and T. Woodall. 2004. Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*. 97–104.

[11] S. Han, H. Mao, and W. Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[12] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy. 2016. MCDNN: An Approximation-Based Execution Framework for Deep Stream Processing Under Resource Constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '16)*. ACM, 123–136.

[13] K. He, G. Gkioxari, P. Dollár, and R. Girshick. 2017. Mask R-CNN. In *2017 IEEE International Conference on Computer Vision (ICCV)*. 2980–2988.

[14] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk. 2015. Session-based recommendations with recurrent neural networks. (2015). arXiv:1511.06939

[15] G. Hinton, O. Vinyals, and J. Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[16] IBM. 2018. *Analyze real-time CCTV images with Convolutional Neural Networks*. Retrieved March 14, 2020 from https://developer.ibm.com/patterns/iot-devicesensor-damage-detection-with-edge-analytics/

[17] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang. 2017. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (Xi'an, China) *(ASPLOS '17)*. ACM, 615–629.

[18] D. Kingma and J. Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun (Eds.).

[19] T. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. 2014. Microsoft coco: Common objects in context. In *European conference on computer vision*. Springer, 740–755.

[20] R. Mullapudi, S. Chen, K. Zhang, D. Ramanan, and K. Fatahalian. 2019. Online model distillation for efficient video inference. In *Proceedings of the IEEE International Conference on Computer Vision*. 3573–3582.

[21] NVIDIA. 2020. NVIDIA Jetson Nano. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/

[22] NVIDIA. 2020. *NVIDIA TensorRT*. Retrieved February 28, 2020 from https://developer.nvidia.com/tensorrt

[23] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035.

[24] I. Radosavovic, P. Dollár, R. Girshick, G. Gkioxari, and K. He. 2018. Data distillation: Towards omni-supervised learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4119–4128.

[25] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–4520.

[26] E. Shelhamer, K. Rakelly, J. Hoffman, and T. Darrell. 2016. Clockwork convnets for video semantic segmentation. In *European Conference on Computer Vision*. Springer, 852–868.

[27] M. Siam, S. Elkerdawy, M. Jagersand, and S. Yogamani. 2017. Deep semantic segmentation for automated driving: Taxonomy, roadmap and challenges. In *2017 IEEE 20th international conference on intelligent transportation systems (ITSC)*. IEEE, 1–8.

[28] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu. 2018. A survey on deep transfer learning. In *International conference on artificial neural networks*. Springer, 270–279.

[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin. 2017. Attention is All You Need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 6000–6010.

[30] Y. Wu, A. Kirillov, F. Massa, W. Lo, and R. Girshick. 2019. Detectron2. https://github.com/facebookresearch/detectron2.

[31] H. Yang, T. Fu, P. Chiang, K. Ho, and C. Lee. 2019. A Distributed Scheme for Accelerating Semantic Video Segmentation on An Embedded Cluster. In *2019 IEEE 37th International Conference on Computer Design (ICCD)*. 73–81.

[32] X. Zhu, Y. Xiong, J. Dai, L. Yuan, and Y. Wei. 2017. Deep feature flow for video recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2349–2358.