ARGOS: Adaptive Recognition and Gated Operation System for Real-time Vision Applications

by

Mohammad Farhadi Bajestani

A Dissertation Presented in Partial Fulfillment
of the Requirement for the Degree
Doctor of Philosophy

Approved April 2022 by the
Graduate Supervisory Committee:

Yezhou Yang, Chair
Sarma Vrudhula
Carole-Jean Wu
Yi Ren

ARIZONA STATE UNIVERSITY

May 2022

ABSTRACT

Deep neural network-based methods have been proved to achieve outstanding performance on object detection and classification tasks. Deep neural networks follow the "deeper model with deeper confidence" belief to gain a higher recognition accuracy. However, reducing these networks' computational costs remains a challenge, which impedes their deployment on embedded devices. For instance, the intersection management of Connected Autonomous Vehicles (CAVs) requires running computationally intensive object recognition algorithms on low-power traffic cameras. This dissertation aims to study the effect of a dynamic hardware and software approach to address this issue. Characteristics of real-world applications can facilitate this dynamic adjustment and reduce the computation. Specifically, this dissertation starts with a dynamic hardware approach that adjusts itself based on the toughness of input and extracts deeper features if needed. Next, an adaptive learning mechanism has been studied that use extracted feature from previous inputs to improve system performance. Finally, a system (ARGOS) was proposed and evaluated that can be run on embedded systems while maintaining the desired accuracy.

This system adopts shallow features at inference time, but it can switch to deep features if the system desires a higher accuracy. To improve the performance, ARGOS distills the temporal knowledge from deep features to the shallow system. Moreover, ARGOS reduces the computation furthermore by focusing on regions of interest. The response time and mean average precision are adopted for the performance evaluation to evaluate the proposed ARGOS system.

To my parents and fiancée

for their love, support, and encouragement.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

Chapter 1

INTRODUCTION

1.1 Problem Statement

Object detection and classification plays a critical role in a variety of embedded applications such as obstacle avoidance (Carrio *et al.*, 2018), detection and tracking (Breuers *et al.*, 2018), object searching (Ye *et al.*, 2018) and intersection management (Khayatian *et al.*, 2020). We have witnessed the great success of Convolutional Neural Networks (CNNs)-based methods in the object detection task during the past decade. This success has led researchers to explore deeper models such as RetinaNet (Lin *et al.*, 2018) or Swin transformer (Liu *et al.*, 2021), which yield high recognition accuracy. The "secret" sauce behind the success of these deeper and deeper CNNs models is the stacking of repetitive layers and increasing the number of model parameters (Chen *et al.*, 2017a). This practice becomes possible while the applications are running in big data centers or infrastructures with high-performance processing capabilities.

However, the disadvantages of this practice are obvious, and the high performance is achieved by the significant growth of the model complexity: stacking up layers and increasing the model parameters of the system, which is computationally expensive and also increases the inference time significantly. Hence, these models are not suitable for real-time and embedded visual processing systems and thus impede their deployment in the era of intelligent robots and autonomous vehicles. The same concerns also lie in the energy conservation and computation limits since deep models require a large number of matrix multiplications, which are time-consuming and energy-demanding for mobile robotic applications.

For instance, the vision-based intersection management (vIM) of CAVs is one of the emerging applications which will become an essential part of cities (Khayatian *et al.*, 2020). A study conducted by American Automobile Association (AAA) shows more than two people are killed every day in the U.S. due to accidents caused by red-light runners (Bomey, 2019), and vIM can significantly reduce it. In vIMs, the processing unit needs to be at the intersection; using cloud computing is not feasible as it requires an extensive network infrastructure that can support the required bandwidth for the cameras; also, the network delay will increase the response time. Moreover, vIM needs to be powered by solar panels in remote regions, limiting the management unit's available energy. Object recognition and tracking is the most energy and computational demanding module in vIM. This problem is aggravated as vIM needs to be accurate and agile in an embedded environment with limited resources.

1.2 Related Work

The last few years in the field of deep learning has laid the foundation for major advancements in visual recognition systems, ranging from object recognition (Lin *et al.*, 2018), action recognition (Lea *et al.*, 2016), to scene recognition (Zhou *et al.*, 2014). Significant improvements in recognition accuracy have allowed a wide range of science fiction ideas to materialize, resulting in economic and societal benefits with AI applications such as autonomous vehicles (Chen *et al.*, 2015a), IoT systems (Tang *et al.*, 2017), industrial robots, and intelligent health care systems (Ravì *et al.*, 2017; Izadyyazdanabadi *et al.*, 2017).

1.2.1 Object Detection

CNN-based object detection methods can be categorized into two groups: two-stage and single-shot detectors. In two-stage techniques such as GP-FRCNN (Amin and Galasso, 2017) and FG-BRNet (Fu *et al.*, 2019), a region proposal network (RPN) localizes regions with a likelihood of object presence, followed by a classification stage. Attempts are made to improve this category of detectors' performance by adaptively adjusting the input image resolution (Chin *et al.*, 2019), processing challenging regions with a separate CNN model (Singh *et al.*, 2018) or using a selective mechanism to reduce the number of proposed regions by the RPN (Kouris *et al.*, 2019). In single-shot approaches such as YOLO (Redmon and Farhadi, 2018), and SpotNet (Perreault *et al.*, 2020), the inference is performed in a single pass by combining region proposal and classification stages. Our work adopts single-shot detectors in experiments considering their higher efficiency. Note, our proposed idea can be applied over two-stage approaches as well such as Faster RCNN.

Image Segmentation, or the task of locating the objects and boundaries in images, is another application for CNN-based methods. Models such as Segnet (Badrinarayanan *et al.*, 2015) have been proposed, which reduce the memory consumption and computation. But research (Saood and Hatem, 2020) has shown that the performance of SegNet deteriorates when there are multiple objects in the scene. Further, DeepLab (Chen *et al.*, 2017b), and PSPNet (Zhao *et al.*, 2017) are improved segmentation models, but these come at the cost of higher computation cost as well. U-Net (Ronneberger *et al.*, 2015) is a CNN that was developed for image segmentation in the biomedical field. It consists of an encoder that extracts the salient features from the image and a decoder part that enables the reconstruction of the binary mask that gives us the location of the objects of interest. Binary segmentation (single class seg-

mentation) has been explored in the past (Putten *et al.*, 2020; Isensee and Maier-Hein, 2020; Gupta *et al.*, 2020) for medical images. We extend the U-Net model (quantized and binarized) to fit different use cases for the task of binary segmentation.

1.2.2 Visual Systems Performance

The increasing number of real-world applications require their corresponding visual recognition engine to not only recognize well but also actively and effectively adjust its computational resources to handle the ever-changing physical world situations that the systems will face. The seminal work of the cascaded classification of Viola and Jones (Viola and Jones, 2001) represents the line of studies on cost-sensitive classification. The essence of their work is to treat classification as a cascaded process that contains control layers deciding the exit points where the system is confident in its current inference. Following a similar line of work, more recently, Li *et al.* (2015); Shen *et al.* (2017) has proposed cascading CNNs structures to reduce the computational cost by reducing the structural complexities of CNNs.

The system resources concerns trigger various approaches, such as using the alignment of memory and SIMD (Single instruction, multiple data) operations to boost matrix operations (Gong *et al.*, 2014), specific hardware (FPGA) solutions (Qiu *et al.*, 2016), and network compression methods (Han *et al.*, 2015a; Kim *et al.*, 2015; Zhang *et al.*, 2016). More recently, studies by Chen *et al.* (2017a) and Hinton *et al.* (2015) proposed transferring the knowledge of deep models to shallow models while maintaining the recognition accuracy.

Another thrust of work has focused on reducing the resources consumption of CNNs (due to expensive computation and memory usage) by compressing the network structures (Ba and Caruana, 2014; Han *et al.*, 2016a, 2015b; Rastegari *et al.*, 2016a). Network pruning is one of well-studied approach which removes unnecessary

connections from CNN model, to gain inference speedup (Han *et al.*, 2016a; Wen *et al.*, 2016; Iandola *et al.*, 2016). Quantizing (Han *et al.*, 2015a) and binarizing (Rastegari *et al.*, 2016a; Courbariaux *et al.*, 2016) are two other methods that have been used to reduce network size and computation load. These methods improve performance at the hardware level by reducing the size of weights at the binary code level. However, the standard GPU implementation still remains challenging for these methods to achieve runtime speedup (Han *et al.*, 2015b). Also, the advantages of these methods over other one-stage methods without the fully connected layers (which is the target layer for network pruning (Han *et al.*, 2015a)) is not clear.

Although these approaches do improve the model efficiency, they ignore the temporal dependencies among the frames from dynamic scenes, which is one of the critical capabilities to maintain high recognition accuracy while being energy-aware.

1.3 Human Visual System

The human visual system (HVS) is the most efficient and effective available visual system. This system has mechanisms such as Visual adaption to improve system overall performance.

Visual adaption involves temporary changes in the human perception system when exposed to intense or new stimulus and by the lingering aftereffects when the stimulus is removed (Webster, 2015). Other studies from (Webster, 2015) show that the visual system adapts to the changes in the environment and this adjustment can happen in a few milliseconds. More specifically, a study from Clifford *et al.* (2007) reveals that the face recognition process happens at a higher level of cognition, and later at the stage of visual encoding we observe that our sensory systems adapt itself to the prevailing environment. This shows that HVS relies heavily on the prior estimation of the objects' appearance distribution to improve the perception capability at the

current time-stamp.

The HVS adaptation happens both in the "low" and "high" level visual features. The human visual system adapts to the distribution of "low-level" visual features such as color, motion, and texture, as well as the "high-level" visual features such as face classification including identity, gender, expression, or ethnicity (Webster, 2015). This adaptation can be both short-term and long-term. For instance, our perception system adapts itself to the general visual features of the environment which we are living in for a long time such as faces and colors (like training a heavy recognition model). Also, it can adapt itself dynamically when the environment changes, for example, moving from the indoor environment to the outdoor (Webster, 2015) (like adapting a shallow model). This adaptation capability is essential for our HVS to perform recognition well and efficient, with low energy consumption.

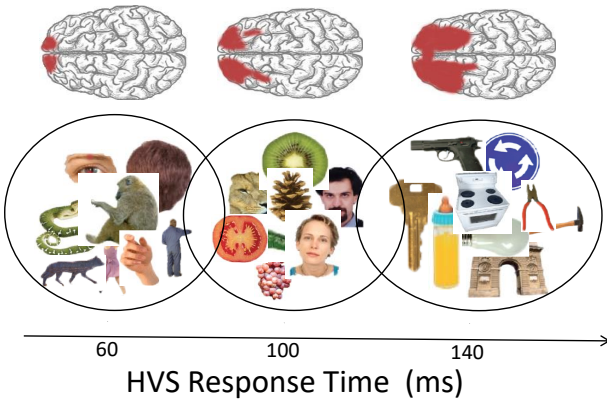

Figure 1.1: Stimulus Materials, Fmri Brain Coverage, and Significant Meg-fmri Fusion Results over Different Objects and Background (Cichy *et al.*, 2016).

In addition to temporal adaptation, the HVS has two stages for conducting visual classification: 1) a shallow primary stage and 2) a decision layer to pick a further processing pathway (Ritchie *et al.*, 2015). The study also supports the theory that the structure of the object representation in the HVS influences the decision layer during visual classification. Results from another research that is conducted in the

field of neuroscience (Cichy *et al.*, 2014) showed that the response time of our HVS given a specific image as the stimuli differs a lot based on the category that the image belongs to. These results again suggest that HVS has a decision system that controls our processing resources assigned for each image. The study by Cichy *et al.* (2016) shows the brain-behavior corresponding to the images from different categories (in Fig. 1.1). From the fMRI imaging, researchers have speculation that for some input images, only a "shallow" part of our HVS is utilized while for other categories they invoke a "deeper" processing.

1.4 Contributions

Inspired by the aforementioned findings, I propose ARGOS, which is capable of running heavy neural networks on embedded systems. I have also deployed ARGOS on a prototype camera which is powered by a solar panel and can be used for traffic monitoring. For designing ARGOS, I have considered both the software and hardware aspect of the system.

In ARGOS, the object detection and classification modules are responsible for detecting and classifying objects in the observing environment. For these modules, I am using the state of the art proposed models that have the capability to adapt to the other ARGOS modules.

The adaptive hardware module controls the resources on the target machine to deliver maximum efficiency. This module changes the computation resource base on the desired accuracy and response time to improve energy efficiency, given that ARGOS is designed for embedded and real-time applications. In chapter 2, I have introduced this module and the challenges ARGOS is facing. Using partial reconfiguration in FPGAs and distributed processing, I am addressing these challenges. A feedback procedure controls the partial reconfiguration. I reported the implementation of the

feedback procedure for determining the path of inference in the CNN based on the confidence level. I have implemented the mechanism on an MPSoC (Pynq-Z1) with an ARM CPU and FPGA.

The temporal adaptation module is responsible for adapting the system to the observing environment. The inspiration for this module comes from visual adaptation in HVS. This module will be responsible for detecting changes in the environment and adapting the whole system to the new changes while maintaining system performance. In chapter 3, I have addressed the challenges and the mechanism for this module. I introduced an end-to-end trainable framework to transfer the temporal knowledge (a.k.a., the perception of the moment) of the oracle model to the student model. I have also designed a novel teacher-bonded loss for knowledge distillation which has a simple structure and performs inferences briskly. I proposed an efficient method to select key frames from the dynamic scene that indicate the right timing to train the student model and improve detection accuracy. I designed and conducted empirical experiments on both the public datasets (the Youtube Object dataset and the Hollywood Scene dataset) as well as on two long videos with multiple scene changes, which validate each of the aforementioned novel design choices, by observing a fast object recognition performance while maintaining high detection accuracy.

In the next stage, I studied the performance and challenges of the temporal knowledge distillation concept in an embedded environment with limited resources. In chapter 4, I have discussed these challenges and different approaches which can address them. Finally, I demonstrate a novel framework for transferring knowledge between two devices, one executing a DNN and the other executing an SHNN. This approach substantially advances edge computing for performing complex and computationally intensive applications. I presented an extensive exploration of various ways in which knowledge transfer can occur and evaluated them in terms of specific, well-defined

metrics. The quality of the detection results depends on when the DNN should be activated for possible knowledge transfer. Toward this, I presented a novel *key frame* selection mechanism that significantly improves the efficiency of the knowledge transfer. The proposed framework for incremental knowledge transfer in object detection is made open-source and has been released for public distribution on Github.

Finally, I studied a novel mechanism to decompose the incoming frames into small independent sub-regions. Hence, ARGOS can focus on regions with a probability of object existence or more complexity for further processing. This mechanism reduces computation time and energy consumption. I designed a mechanism (based on shallow analysis or binary segmentation) that can efficiently propose regions of Interest (RoIs). I have also extended the ARGOS concept to a variety of DNNs, such as transformers and online knowledge distilation. Finally, A full-stack developed system on-device with extensive experimentation. I have tested the prototype for the vIM application.

1.5 Dissertation Outline

The rest of this dissertation is organized as follows. In chapter 2, I have shown the potential and challenges of adaptive hardware. The architecture and mechanism which can be used to facilitate this adaption have also been discussed. In chapter 3, the temporal adaption mechanism, which is inherited from the human visual system, has been described that can reduce the number of hardware adaptions and improve performance. Moreover, I have studied the consequences of this approach on system accuracy. In chapter 4, I showed how the temporal adaptation could be transferred to an embedded environment. I studied different approaches for this adaptation and the communication medium effect on this adaptation. In chapter 5, I merged the previous modules and combined them with a mechanism to reduce to input deployed for the vIM application. This mechanism can reduce the computation by focusing on

the regions of interest. Finally, we concluded this dissertation in chapter 6.

Chapter 2

ADAPTIVE HARDWARE

2.1 Introduction

The ultimate goal of the ARGOS is to be able to apply computer vision algorithm on embedded devices in real-world applications. To run the computer vision applications, we should manage the hardware resources due to low energy constraints and limited computing resources on these devices. We should achieve the system desire accuracy and response time while maintaining energy. We can adapt distributed computing to answer the limited computation (Eshratifar and Pedram, 2018) in embedded devices. However, this approach can significantly increase the system response time. In the following, I have summarized the challenges we are facing on the hardware side.

- Reduce the system response time.
- Manage energy consumption.
- Maintaining the accuracy while decreasing the response time.

2.2 Related Work

In the last few years, many designs and mechanisms proposed to optimize neural networks for running over embedded networks. Cai *et al.* (2019) proposed using neural network search to design networks based on hardware architecture. Eshratifar and Pedram (2018) proposed a distributed structure to offload part of network

computation on cloud resources. Although this works can respond to some concerns, they can not provide the desired accuracy and response time in all scenarios.

Another thrust of work has focused on reducing the resource consumption of CNNs or other types of neural networks through various techniques of compressing the network structures (Ba and Caruana, 2014; Han *et al.*, 2016a, 2015b; Rastegari *et al.*, 2016a). Network pruning is one of the well-studied approaches which removes unnecessary nodes and edges from network, to compress model and gaining inference speedup (Han *et al.*, 2016a; Wen *et al.*, 2016; Iandola *et al.*, 2016). However, Han *et al.* (2015b) pointed out that using the standard GPU implementation, the speedup is hard to achieve due to the lack of high degrees of exploitable regularity and computation intensity in the resulting network with sparse connections.

The use of adaptive structures is a relatively newer approach which decides how to further process the image (Shen *et al.*, 2017; Zhou *et al.*, 2017; Teerapittayanon *et al.*, 2016; Bengio *et al.*, 2015). Teerapittayanon *et al.* (2016) proposed an adaptive model to allow early exit based on the entropy of model output which is called Branchy-Net. By adding sub-outputs to the model, Branchy-Net checks the entropy of model output and if the entropy is low enough, terminates the procedure. By doing this, Branchy-Net achieved 2x speedups at the inference time (Teerapittayanon *et al.*, 2016). However, Branchy-Net spends a considerable amount of time to evaluate the early output (Bolukbasi *et al.*, 2017); it does not a have clear procedure to select the location of early branches, and it is changing the structure of original model to have an early exit. In response to the mentioned issue, Bolukbasi *et al.* (2017) proposed an adaptive method which adopts the Branchy-Net idea and stacks several models such as AlexNet (Krizhevsky *et al.*, 2013) and ResNet (He *et al.*, 2016a). This model still suffers from the overhead time of evaluating the model's early output. Another study in (Wang *et al.*, 2018b), proposed a method based on a decision gate to skip

some of the blocks in ResNet structure. The decision gates include convolution and fully connected layers which are trained using reinforcement learning. These decision gates are not suitable for shallow CNN models such as ResNet-18 (He *et al.*, 2016b).

The implementation of CNNs on FPGAs has been studied from the literature to certain extent. More specifically, *BinaryEye* in (Jokic *et al.*, 2018) has presented an implementation of binary neural networks on FPGA. The presented implementation can be used in IoT and distributed systems where the stream of images for a camera needs to be processed. A framework called *FINN* has been also presented in (Umuroglu *et al.*, 2017) for the inference of binarized neural networks. The mentioned implementation does not adopt the partial reconfiguration to address the limitation of resources on FPGA.

Dynamic partial reconfiguration has been done in the relevant literature in (Al Kadi *et al.*, 2013; Kästner *et al.*, 2018). In (Al Kadi *et al.*, 2013), the authors have implemented the reconfiguration steps in a Zynq 7000 FPGA but do not implement CNN architectures. Dynamic reconfiguration has been done in (Kästner *et al.*, 2018) for the CNNs on the Pynq board. In the mentioned work, they have stated that the implementation of CNN using reconfiguration at each layer is expensive.

2.3 Proposed approach

To address concerns in 2.1, I have implemented the idea of adaptive switching between shallow and deep networks on FPGA platform using partial reconfiguration to reduce the amount of needed computation. The confidence level was observed to be the most efficient factor to switch in comparison with the methods presented in Skip-Net (Wang *et al.*, 2018b) and Branchy-Net (Teerapittayanon *et al.*, 2016).

2.3.1 Adaptive and Hierarchical CNNs

The key module of my proposed Adaptive and Hierarchical convolutional neural networks (AH-CNN) model is a feedback procedure which is designed to comprehensively evaluate the classification procedure. More specifically, AH-CNN consists of three parts: 1) a **shallow part** which is a light-weight CNN model; 2) a **decision layer** which evaluates shallow part's performance and makes a decision; and 3) a **deep part** which is a deep CNN with a high inference accuracy. The overall objective of my dynamic system is to obtain the highest possible recognition accuracy during critical time instances while maintaining a satisfiable performance using the shallow part during non-critical moments. Following this intuition, I put forward a mechanism with a combination of a shallow model, feedback procedure and a deep model, which has a flexible structure at the same time. This mechanism can achieve the same high recognition accuracy as other very deep networks by partially reconfiguring the hardware structure. Thus, an intelligent agent equipped with the AH-CNN can adaptively adjust its model structure to maintain a balance between the expected classification accuracy and the model complexity. This procedure can be applied repetitively and has several decision layers. In the following section, I will describe the details of the AH-CNN architecture.

2.3.2 AH-CNN Architecture

The authors in (Zeiler and Fergus, 2014) showed that the preceding layers in deep neural networks respond to class-agnostic low-level features, while the rear layers extract more specific high-level features. Objects of certain categories can be classified solely by the low-level features but for the images of other categories, we need more specific high-level features, and deeper layers are needed to extract them. Thus,

I design my architecture to have three modules: the shallow part, the deep part and a decision layer. Hence, the proposed AH-CNN with a design of an adaptive and hierarchical structure, can yield different behaviors based on the input image characteristics. I will describe the three mentioned modules in the following.

Shallow Part: In this work, the FPGA is loaded with the shallow part first. This part can be applied to the input tensor without any reconfiguration cost and classifies all input images and it outputs two results: 1) a predicted label $y = j$ and 2) a confidence value ($P(y = j|X_i) = softmax(z_j) = exp(z_j)/\sum_k exp(z_k)$, where z is the output of fully connected layer over the input image X_i) which will be later used in the feedback procedure.

Deep Part: This part is the next group of convolution layers which should be loaded on FPGA. Due to the transfer and configuration time, loading the new part on the FPGA is expensive. This group of convolution layers is responsible to extract more specific high-level features and detect the images which are misclassified by the shallow part. This part will be applied over the output of the last convolution in the shallow part to reach higher confidence. **Decision Layer:** This part of AH-CNN takes the shallow part's outputs and makes a decision to whether activate the deep part, or simply terminate further processing and take the shallow part's result as the overall model output. This layer has a feedback procedure to make the network behavior decision by evaluating the shallow part.

To this end, the decision layer currently yields a binary behavior based on three factors: 1) the confidence value from the shallow part; 2) the priority of the object classes; 3) the overall expected classification accuracy (which is obtained by validating the model over the data set). The binary behavior either activates the deep part or takes the shallow part's classification output as the overall model's output.

Algorithm 1 shows the AH-CNN processing procedure in the inference phase.

Algorithm 1 AH-CNN: Inference Phase

Require: Input image X_i , Desired accuracy Λ, Number of early branches N_i, High priority classes S_{HP}.

```
while X_i do
  while N_i do
    Assign proper Γ based on Λ
    β, ShOutput ← ForwardPropagate(X_i, Shallow)
    if S_HP appear in ShOutput Top-n then
      Γ = Γ + Θ
    end if
    if β <= Γ then
      Load deep part on FPGA
      Output ← ForwardPropagate((ShOutput, Deep)
    else
      Output ← ShOutput
    end if
  end while
end while
```

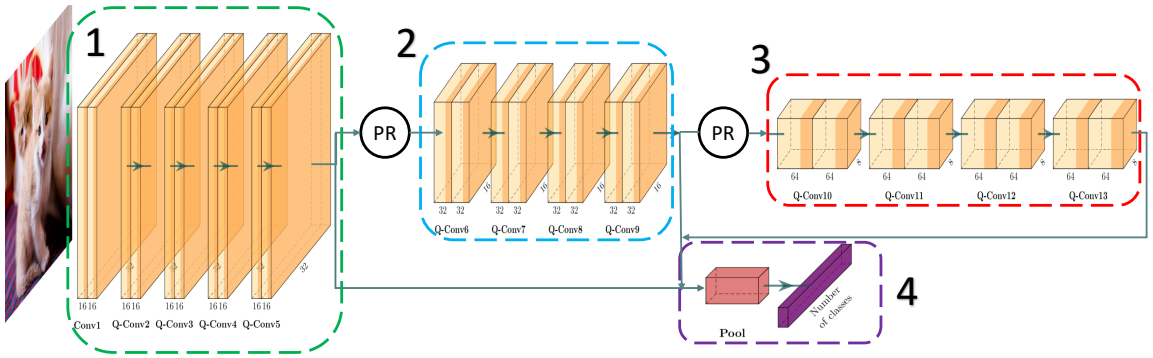

Figure 2.1: The Scheme of Cnn Implementation on Fpga Using Dynamic Reconfiguration and Adaptive Feedback. The Adaptive Feedback Makes the Decision to Classify the Image or Apply the next Stack of Convolution Layers Based on the Output Confidences Computed from Each Part.

The decision layer first checks the top-n classification results from the shallow part's classification vector. If a label from the high priority set (S_{HP}) exists, there is a higher probability that the input needs further processing. Next, the decision layer checks the current expected classification accuracy, which will affect the fraction of all the input images that need further processing. Finally, the model checks the shallow part's Confidence value. The interpretation of the confidence value yields a feedback

procedure. The priority of the object classes and the overall expected classification accuracy are then considered to tune a threshold value to compare with the confidence value, which I refer to it as the trigger point later.

The most critical element of the feedback procedure of AH-CNN is the trigger point Γ. After feed-forwarding each image over the shallow part, the decision layer gets the confidence value β and compares it with the assigned threshold Γ. If β does not reach Γ, it means that the shallow part has less confidence than my system's tolerance over the input image and further processing is needed to gain a higher expected accuracy. As a consequence, the decision layer load and activates the deep part. The value of the trigger point can be actively adapted according to the real-world situations. In cases that I do not need a high accuracy, I can decrease the trigger point value. In cases that the member of S_{HP} appear in the top-n outputs, I can increase the trigger value (Γ) by Θ to expect a higher classification accuracy over that image. The trigger point makes my model innately adaptive. I discuss how to set a proper trigger point as well as its range in Section. 2.3.6.

2.3.3 Implementation on FPGA

The overall scheme of implementation on the FPGA is depicted in Figure 2.1. The convolution layers in the CNN are based on the ResNet CNN structure. The whole CNN is divided into three parts which are numbered in the figure. The output of each part can be used as the input for the pooling layer in part 4. There is a Partial Reconfiguration unit labeled as *PR* which changes the bitstream file on the FPGA when necessary. The reason for partial reconfiguration is to save the LUT area on the FPGA and address the limitation of computational resources.

In order to implement the CNN on FPGA, a quantized version of CNN has been used which is popular in the FPGA community (Umuroglu *et al.*, 2017). In this

network, the weights are binary and the activation data are five bits (quantized bits). Even using this quantization method and binary values, an acceptable accuracy of classification can be obtained which is shown in the experimental result section.

Batch processing has been used to improve the overall throughput of the system. During batch processing, the reconfiguration overhead of changing the bitstream files would be considered for all the images that are going to be processed in the network. Therefore, the overhead of reconfiguration would be negligible when calculating the inference time for one image on average.

2.3.4 Training Phase

Both the shallow and the deep part aim to classify images with the best possible performance that can be achieved individually. Consequently, the feedback procedure should not have any influence over the shallow part's classification performance. I train both the deep part and the shallow part using the stochastic gradient mini-batch (Dean *et al.*, 2012). Also, the mean and range of trigger point value are needed to be learned from the training data. In the following sections, I first introduce the overall model learning procedure in Section. 2.3.5, and then report my training details in Section. 2.3.6.

2.3.5 Learning Procedure

In the first stage, all parts are trained jointly over training set S_T and validated over validation set S_V. In each epoch, the accuracy of all parts are evaluated over the validation. The model with the highest accuracy over the deepest part will be selected as the best model due to reaching the best possible accuracy at critical inference time.

Identifying the trigger point: Following the aforementioned design, the shallow part after feed-forwarding each input image has a confidence value over the output

belief vector. To have an evaluation over this value and its range, I feed all images from S_T into shallow part and collect the confidence values. The calculated mean C_{Mean} and the standard deviation C_{Std} over these values are used to control the expected classification accuracy of the AH-CNN.

2.3.6 Model Training Details

Initializing: I first adopt the ResNet-18 model as the base model, where each of the blocks in this model is considered as a separate classification module. I added a pooling and a fully connected layer for each part. Xaviar initialization (Glorot and Bengio, 2010) is used for having proper initial weights to propagate the signals precisely.

Defining the loss function: For a classification task, the cross entropy is mostly used as loss function. Here, We have several parts which get their input from previous layer and have independent classification layer output. Consequently, these parts should be trained jointly. The objective function can be formulated as

$$L(\hat{y}, y; \theta) = \sum_N L(\hat{y}_n, y; \theta),$$

where

$$L(\hat{y}, y_n; \theta) = -\frac{1}{\zeta_{S_T}} \sum_k y_n^k \log f(x_k; \theta),$$

and N denotes the total number of classification modules, x_k the input images, ζ the set of all possible labels and $f(\theta)$ denotes the whole model.

2.3.7 Experiments

The theoretical framework I have presented suggests two hypotheses that deserve empirical tests: 1) AH-CNN can perform visual classification with much higher effi-

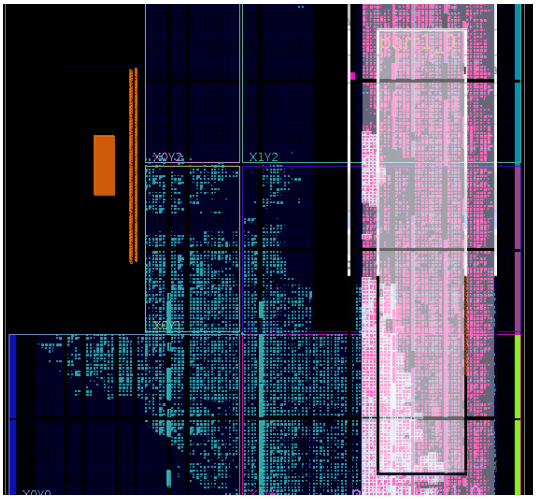

Figure 2.2: Layout of the Reconfigurable Design.

ciency while maintaining the accuracy; and 2) Deep CNN models can be executed on a resource-constrained FPGA using partial reconfiguration.

To validate these two hypotheses, I implement AH-CNN on Xilinx Zynq-7000 and evaluate on the CIFAR-10, CIFAR-100 (Krizhevsky and Hinton, 2009) and SVHN (Netzer *et al.*, 2011) datasets. I implemented the AH-CNN as described in Sec. 2.3.1 where all convolution parts were implemented as separate hardware IP cores. I utilize Vivado HLS to synthesis the IP cores. The training procedure was performed using the PyTorch framework.

2.3.8 Implementation

I select the PYNQ-Z1 to perform my evaluations. This board consists of a Xilinx Zynq-7000 ZC7020 and a dual-core ARM A9 processor. Images were loaded to my convolution IP cores through a Direct Memory Access (DMA) IP core.

I adopt the Resnet-18 (He *et al.*, 2016a) as the base model. Due to limited available LUTs on this board, the network was broken into three parts. All parts consist of a group of convolution layers, pooling and fully connected layer. To reduce

the reconfiguration time, I remove the last pooling and fully connected layer and create a new part which will be shared. Figure 2.1 shows an overview of my model. Part 1 is the shallowest model of this architecture. Parts 2-3 are the deeper blocks for extracting more features. Part 4 is the common one among all. Table 2.1 shows the resources needed for each part and total available resources on FPGA.

	Part 1	**Part 2**	**Part 3**	**Part 4**	**Total**
BRAM	81	91	96	31	280
DSP	120	96	96	24	220
FF	15672	16647	34069	9908	106400

Table 2.1: Available Resources on the Zynq Xc7z020, in Comparison to Used Resources by Convolution Parts.

As shown previously, the total hardware resources needed for the whole architecture is more than available resource over the target device. Moreover, there are shared modules over all convolution parts such as Part 4, DMA, etc. Consequently, I applied Dynamic Partial Reconfiguration in order to reduce the reconfiguration time by just changing the convolution parts and keeping the shared modules. Fig. 2.2 shows the layout of my implementation. The reconfigurable area is shown by purple and the fixed ports on the FPGA by white.

The resulting partial parts have all the same size of 2.4 MB and the size of main bitstream is 4 MB.

Training: The training part was carried out using PyTorch framework. I implemented special quantized convolution layer and fully connected layer with 1-bit weight and 5-bit activation. The initial learning rate is set to be 0.01 and it was decreased by a factor of 10 in every 20 epochs. Training continues until 100 epochs with a mini-batch size of 256.

Feedback Evaluation: The aforementioned procedure in section. 2.3.6 is followed to estimate the confidence value. The mean and the standard deviation of all

the confidence values were achieved after the various parts were collected over S_T.

2.3.9 Overall Evaluation

I choose the CIFAR10, CIFAR-100, and SVHN validation sets in the overall AH-CNN model testing. Here, I evaluate the partial reconfiguration approach. Also, I compare three selection methods: 1) my proposed feedback procedure; 2) SkipNet method (Wang *et al.*, 2018b); and 3) an entropy-based method (Bolukbasi *et al.*, 2017).

Bitstream	FPGA Config Time	FPGA Execution Time	CPU Execution Time	FLOPS
Part 1	38-42 ms	2 ms	98 ms	10.24M
Part 2	38-42 ms	2 ms	57 ms	8.6M
Part 3	38-42 ms	2 ms	49 ms	8.5M

Table 2.2: Performance Evaluation on Different Parts of the Design.

Partial Reconfiguration: I have three accelerator IPs to reconfigure which are connected to the ARM processor through AXI interface, clocked at 100 MHz. The AXI channel and partial reconfiguration module is controlled by a Python script. I have also implemented a CPU version of AH-CNN architecture which runs on an ARM chip at 666 MHz. Table 2.2 shows the measurements of partial reconfiguration, FPGA and CPU execution time. As the reconfiguration region is same for all IPs, The reconfiguration time is always the same. By using batch processing (batch=512), the throughput of my system is ≈160 image per second while applying all parts to the images. This is 32 times faster than the CPU implementation.

Table 2.3 shows the accuracy that can be achieved by applying each IP of convolutions to the input stream. It is clear that the system can reach to the higher accuracy by extracting more feature using deeper layers. Also, a significant portion

	CIFAR10	CIFAR100		SVHN
		Top1	Top5	
Part 1	70.95	42.26	72.14	80.35
Part 2	80.57	52.23	80.25	91.24
Part 3	86.27	56.60	83.46	94.62

Table 2.3: Top-1 Accuracy of the HLS Optimized IP-cores.

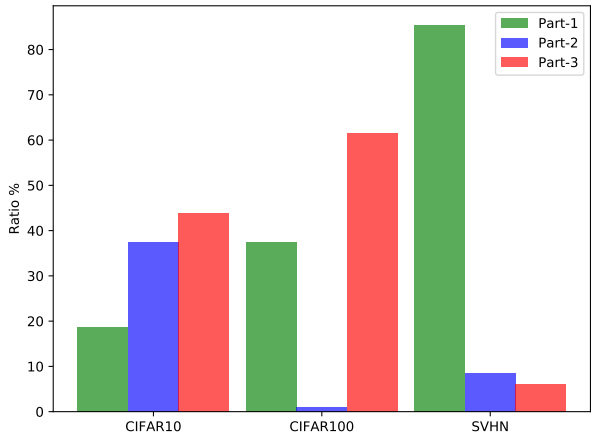

Figure 2.3: The Stop Ratio for Each Part on Cifar-10, Cifar-100, and SVHN Dataset.

of images can be classified correctly without using deep layers.

Feedback Procedure: Initially, I explore the trigger point by collecting the confidence of each AH-CNN branch. AH-CNN model achieves 85.4%, 55.4%, 94.2% Top-1 validation accuracy over CIFAR10, CIFAR-100, and SVHN respectively. In Fig. 2.3, I also report the portion of images classified by each branch. Due to the simplicity of the feedback procedure, this method has the lowest overhead.

SkipNet (Wang *et al.*, 2018b): In this method, instead of selecting images by my feedback procedure, decision layer selects images using a gate consisting of convolution and fully connected layers. I adopt two different gates and two training methods proposed by Wang *et al.* (2018b) to evaluate my method. These gates show desirable performance over large CNN models. However they do not have the same performance over models such as ResNet-18 or ResNet-38. For each decision, one or two convolution layers and a fully connected layer should be applied to the stream.

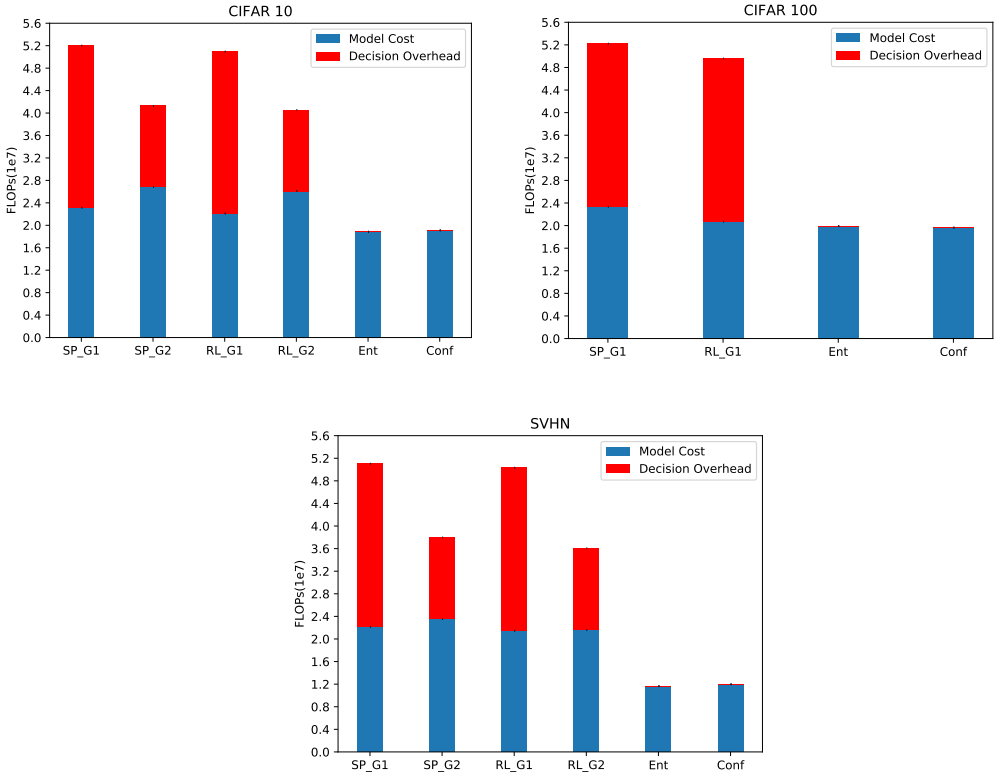

Figure 2.4: Computation Reduction of Entropy (Ent), Confidence (Conf), Skipnet+SP (Sp) and Skipnet+hrl+sp (Rl) with Feed-forward Gates (G1 Has Two Convolution Layers, G2 Has One Convolution Layer) While Preserving the Full Network Accuracy. The Computation Cost Includes the Computation of Decision Method. I Am Able to Reduce the Computation Costs by ≈30%, ≈27% and ≈57% on the Cifar-10, Cifar-100, and Svhn Data Using Confidence Decision Method Compared to the Base Model. Since the Feed-forward Gates Are More Expensive, Skipnet Is Not a Suitable Method for the Scope of the Study.

Entropy Selection (Bolukbasi *et al.*, 2017): This method uses the entropy of the shallow part's output to decide whether the input image needs further processing or not (Bolukbasi *et al.*, 2017). The work (Bolukbasi *et al.*, 2017) implemented two variants: two-stacked model (AlexNet (Krizhevsky *et al.*, 2013) and -50 (He *et al.*, 2016a) and three-stacked model (AlexNet, GoogleNet (Szegedy *et al.*, 2015) and ResNet-50). Due to calculating the entropy of the output vector at each branch, this method is more expensive than the feedback procedure.

Fig. 2.4 depicts the computation reduction by applying the different decision procedures. I observe that by just considering the confidence, the model outperforms the SkipNet gates. SkipNet gates not only are so expensive but also are not as successful as other methods in my case study. The confidence and entropy selection have the same results however the confidence method has less computation cost. The confidence selection method decreased the computation to 69.8%, 71.8%, 43.8% of the base model in CIFAR-10, CIFAR-100, and SVHN respectively. Also, the throughput of model reaches to 268, 217, and 408 images per second.

2.4 Conclusion

This chapter proposed a new approach to running heavy neural networks on FPGAs with constrained resources. I stacked various shallow and deep models yielding an adaptive and hierarchical structure for quantized neural networks. I conducted experiments on CIFAR-10, CIFAR-100, and SVHN and empirically validated that AH-CNN maintains a similarly low inference time as the shallow models while achieving the high recognition accuracy of the deep model on image classification tasks. The flexible nature of this hierarchical method makes it suitable for applications that need adaptive behavior towards dynamic priority change over object categories, such as an agent with active perception. Although this mechanism can improve the system performance, the number of hardware adaptions can still affect inference time. The next chapter introduces a concept that can reduce hardware adaptation by learning from previous observations.

Chapter 3

TEMPORAL ADAPTATION

3.1 Introduction

As illustrated in the introduction section, the Human visual system (HVS) adapts to "high" and "low" level features to reduce resource utilization and improves the response time Webster (2015). We can also use the prior estimation of the objects' appearance distribution to improve the system performance. For clarity, in autonomous car application, we are not expecting to observe "ship" in the scene, so we can shift the system domain of knowledge to improve the accuracy over objects which have a higher chance of being seen. However, this effort can cause that system to lose its generality over other objects. Consequently, we are trying to adapt the system to the observing environment while keeping the system generality. This approach can help us to use a shallow model (which is adapted to the environment) for observing the environment and use a deeper model for new objects and adaptation.

To summarize problems:

- Temporal adaptation: we need a method to distill temporal knowledge form the shallow model to deep model.
- Keep the system generality: ARGOS should maintain the state of the art vision model accuracy.
- Cost-effective: training a neural network is an expensive manner, this procedure should be cost-effective at inference time.

3.2 Related Work

Domain Adaptation: Object detection in the real world still needs to address challenges such as low image quality, large variance in the backgrounds, illumination variation, etc. These could lead to a significant domain shift between the training, validation and the test data. Consequently, the field of domain adaptation has been widely studied in image classification Tzeng *et al.* (2014); Lu *et al.* (2017) and object detection Chen *et al.* (2018); Dai and Van Gool (2018) tasks. These methods improves accuracy on well-known bench-marking datasets. Nevertheless, they typically adopt an offline domain adaptation procedure and do not concern with domain-change during the inference stage.

Knowledge Distillation: Knowledge distillation is another approach to boost accuracy in CNNs. Under the knowledge distillation setting, an ensemble of CNN models or a very deep model will serve as the teacher model, which transfers its knowledge to the student model (shallow model). Hinton et al.Hinton *et al.* (2015) proposed a method to apply teacher prediction as a "soft-label" and distills teacher classifier's knowledge to the student. Moreover, they proposed a temperature cross entropy instead of $L2$ distance as the loss function. Romero et al.Romero *et al.* (2014) proposed a so-called "hint" procedure to guide the training of the student model. There are also other approaches to distill knowledge between different domains such as from RGB to depth images Gupta *et al.* (2016); Su and Maji (2016).

Knowledge distillation has been also applied to the object detection task. Chen et al.Chen *et al.* (2017a) proposed a method which adopts all of the soft labeling (labels generated by the teacher), the hard labeling (the ground truth) and the hint procedure to transfer knowledge from the teacher with deep feature extractor to the student with a shallow feature extractor. They adopt a two-stage method (FasterRCNNRen *et al.*

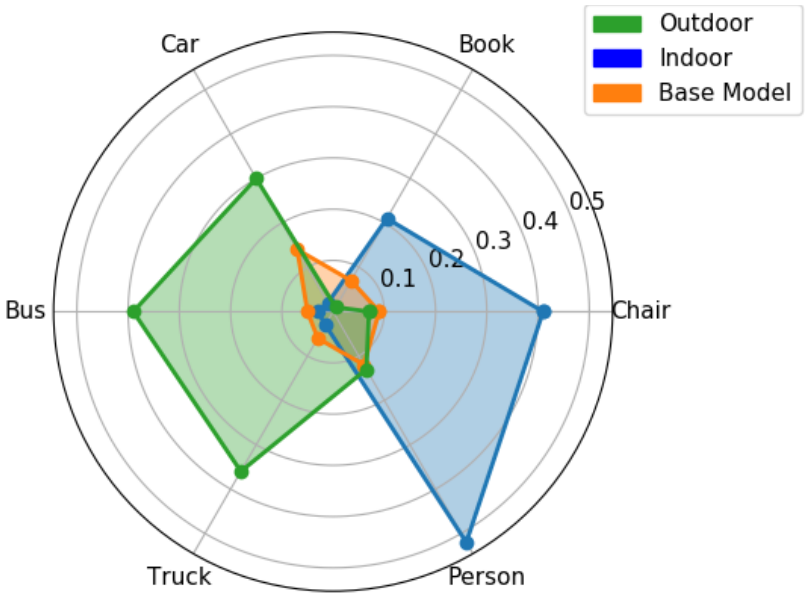

Figure 3.1: An Illustration of the TKD Model's Actual Performance: F-1 Score Distribution over Example Object Categories in Different Environments Using TKD.

(2015)) in their system. Mehta et al.Mehta and Ozturk (2018) applied same procedure to one stage method (Tiny-Yolo v2).

Mullapudi et al.Mullapudi *et al.* (2018) proposed an online model distillation for efficient segmentation. They adopt a light CNN model as student and a heavy model as teacher. At the inference time, the student model is trained periodically using the teacher knowledge. However, the naive usage of a fixed period may not be efficient in their approach. Moreover, it may not handle never seen objects in the scene since they have been observed in the middle of a period.

3.3 Proposed approach

In this section, I am introducing my complected work to address challenges in Sec 3.1. I design a framework that utilizes the knowledge distillation techniques to transfers temporal knowledge from the heavy model to a light model, named TKD. The TKD framework boosts visual processing efficiency while maintaining the heavy

model's (also called the oracle model) performance. Figure 3.1 illustrates the overall goal of this work. In this figure, I show how TKD improves recognition accuracy over different scenes, compared to the oracle model which I assume to be a perfect model. Also, I have the baseline model which is a tiny model with low accuracy compared to oracle recognition due to a much smaller number of parameters. TKD achieves higher accuracy by adapting itself to the observed environment. In the case of an indoor scene, the TKD recognition accuracy improves significantly over objects which are more probable to be observed inside a building. In the outdoor case, TKD recognition accuracy improves over the objects such as a car, bus, and truck which are more probable to be observed outside. In other words, for a similar amount of model parameters as the baseline tiny model, the TKD achieves much better performance over the more probable objects by dynamically learning from the oracle model.

3.3.1 Temporal Knowledge Distillation

Conventional use of knowledge distillation has been proposed for training CNNs based classification models. In these models, I have a dataset $(x_i, y_i), i = 1, 2, ..., n$ where x_i and y_i are input images and the class labels. The student model is trained to optimize the following general loss function:

$$\begin{aligned} &O_s = Student(x); O_t = Teacher(x) \\ &L(O_s, (y, O_t)) = \beta L_{gt}(O_s, y) + (1 - \beta) L_t(O_s, O_t), \end{aligned} \tag{3.1}$$

where L_t is the loss using teacher output (O_t) and L_{gt} is the loss using ground truth y Mehta and Ozturk (2018); Chen *et al.* (2017a); Hinton *et al.* (2015). Here, β is a modulation factor.

In addition to the classification task, object detection also could benefit from knowledge distillation procedure. However, it's not as straightforward as the classification task. Most notably, the teacher model's output may yield misleading guidance

to the student modelChen *et al.* (2017a). The teacher regression result can be contradictory to the ground truth labels, also the output from the teacher regression module is unbounded. To address these issues, Chen *et al.* (2017a) proposed a procedure to only adopt teacher's output at beneficial times. For a one-stage object detection setting, Mehta and Ozturk (2018) optimized the student model with a similar loss function to Eq. 3.1.

In this work, I propose a novel and bio-inspired way of adopting the teacher model's knowledge. Namely, temporally estimating the expectation of object labels, their sizes and shapes based on the previous observed frames or $E[y_i|\alpha_1, \alpha_2, ..., \alpha_{i-1}]$ where y_i is our objects label and α our observations. This expectation changes in time by camera or objects movements, and/or the changing of the field of view. Here, I utilize this extracted knowledge to improve object detection performance. Unlike the previous work such as Mehta and Ozturk (2018); Chen *et al.* (2017a), I am not aiming to improve the feature extractor and/or the general knowledge of the student model. I optimize the decoder inside the student model to adapt it towards the current environment. It is done by increasing the likelihood of objects which are more frequently found from the previous observations. Since the model requires the online training during the inference stage, it should be able to address the following challenges:

1. Training is a time consuming procedure, running it at inference stage will hurt model efficiency;
2. Selecting the key frames which the student model needs to be adapted;
3. Objects with low appearance probability may not be detected by the student model after adaptation;
4. The Oracle model still will introduce noise at locations where there are no objects. Simply training the student model with noisy oracle output decreases

the accuracy.

In the following, I will introduce my approach to address these challenges respectively.

3.3.2 TKD Approach

In this work, I adopt Yolo-v3 (as teacher) and Tiny-Yolo v3 (as student) Redmon and Farhadi (2018) as the base object detection methods. These two models are one-stage object detection models. In both models, object detection is conducted at various layers. The middle layers are used to detect large objects and the last layers to detect small objects. Studies Redmon and Farhadi (2018), Mullapudi *et al.* (2018) and Lin *et al.* (2018) showed that this strategy successfully improves the object detection accuracy with a significant edge.

The overall objective of proposed system is to estimate the expectation of object labels, their sizes and shapes on the temporal domain and to improve the performance of the student model. Following this intuition, I put forward a mechanism with a combination of an oracle model (which I consider it as the best possible model) and a student model (which is fast but has considerably lower accuracy compared to the oracle). TKD transfers the temporal knowledge of the oracle model to the student model at the inference time. By transferring this knowledge, the student model adapts itself to the current environment or scene. Without loss of generality, I select the Yolo-v3 object detection model as the oracle model due to its reliable and dominating performance compared with other one-stage methods. I select the Tiny-Yolo detection model Redmon and Farhadi (2018) as the student model due to its high base frame rate and having a similar model structure with the Yolo-v3.

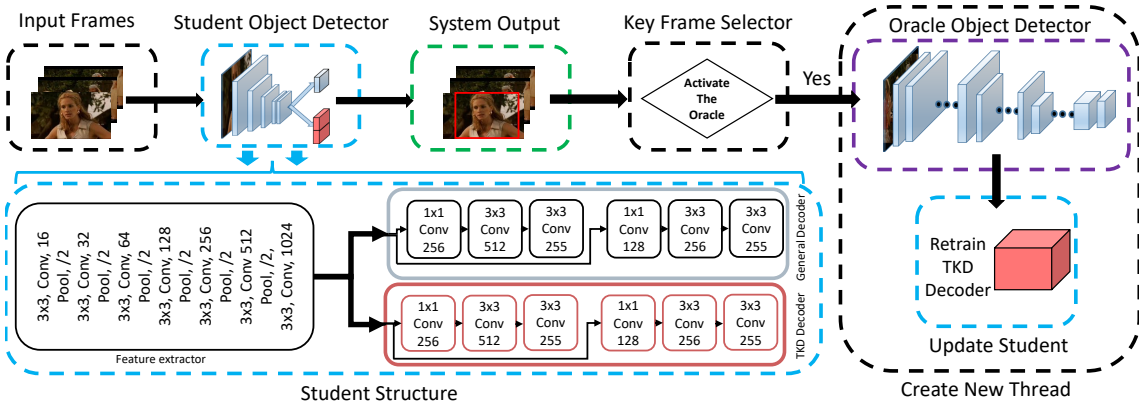

Figure 3.2: An Overview of Tkd (Temporal Knowledge Distillation): A Low-cost Student Model Is Tasked to Detect Objects in the Main Thread. To Retain High Accuracy, a Key Frame Selector Decides to Activate the Oracle Model and Adapt the Student over the Environment. Since the Execution of the Oracle Model and Retraining the Student Model Occurs in Separate Thread, It Does Not Have Significant Effect on the Inference Latency.

3.3.3 The TKD Architecture

I show the overall framework in Figure 4.2. In the student model, I include two decoders as the TKD decoder and the general decoder. Then, the pre-trained Yolo-v3Redmon and Farhadi (2018) is adopted as the oracle. I run the Oracle model with the input image and the weights of student's TKD decoders get updates at specific frames from the oracle model's result. Finally, I design a decision procedure using an LSTM model, to generate the signals that indicate the right timing to use the Oracle knowledge.

Specifically, I train Tiny-Yolo with general decoder over the COCO datasetLin *et al.* (2014). The design of Tiny-Yolo has two general decoders to improve the accuracy over different object sizes. I first make a copy of the general decoders bounded together as TKD decoder. The TKD decoder is updated during the inference stage. I only update the last three layers of Tiny-Yolo and treat it as the decoder, scince it yields enough performance in practice. I keep the general decoder from Tiny-

Yolo together with the TKD decoder to make the final detection. TKD decoder and general decoder are executed in two parallel threads which do not increase the latency significantly. This will preserve the chance of detecting viable objects addressing challenge (3) in Sec. 3.3.1.

3.3.4 Distillation Loss

Before describing the proposed distillation loss, I provide a brief overview of the other distillation loss functions. First, Chen et al. Chen *et al.* (2017a) proposed a combination of hint procedure and weighted loss function. They generate boxes and labels using both the student and the teacher model, then calculate two loss values comparing the teacher's output and the ground truth. At the end, they sum up the weighted loss values. If the student model outperforms the teacher model, they continue training only using the ground-truth supervision. More recently, Mehta et al. Mehta and Ozturk (2018) applied the similar procedure to the one-stage object detection models (Tiny-Yolo v2 with some modification). They generate bounding boxes and labels, and apply Non-Maximum Suppression (NMS) to these boxes and then follow the loss function to optimize the student model. The loss is defined in the following equation:

$$\begin{aligned} L_{final} = L^C_{bb}(b_i^{gt}, \hat{b_i}, b_i^T, o_i^T) + L^C_{cl}(p_i^{gt}, \hat{p_i}, p_i^T, o_i^T) \\ + L^C_{obj}(o_i^{gt}, \hat{o_i}, o_i^T), \end{aligned} \tag{3.2}$$

where $L^C_{bb}, L^C_{cl}, L^C_{obj}$ are objectness loss, classification loss and regression loss which are calculated using both ground truth and the teacher output. Also, $\hat{b_i}, \hat{p_i}, \hat{o_i}$ are bounding box coordinates, class probability and objectness of the the student model. $b_i^{gt}, p_i^{gt}, o_i^{gt}$ and b_i^T, p_i^T, o_i^T are values derived from ground truth and the teacher model output.

In my study of the Yolo-v3 and Tiny-Yolo models, I noticed that the detection

layer is the most computationally expensive part. In this layer, several processes are done (sorting, applying softmax to classification cells, removing low confidence boxes, etc.) to produce bounding boxes and then applying NMS to these boxes. These processes are computationally slow due to its multiple steps of processing, and also running over CPU by implementation. Consequently, directly adopting these loss functions will be also computationally expensive during the inference stage.

With this observation, I adopt the mean square error (MSE) between the tensors generated by the student decoder and the oracle decoder, which should be the fastest method. However, the side effects are also notorious. The oracle model generates noises over some parts of frame which has no object existences; hence directly forcing the student model to retrain will hurt its performance.

Another approach could be calculating the MSE between the tensor cells which have a high confidence of object existence. But, the approach will hurt the student's recognition accuracy too. By applying this loss function, the student model tends to generate redundant detection boxes which yields a larger number of false positives.

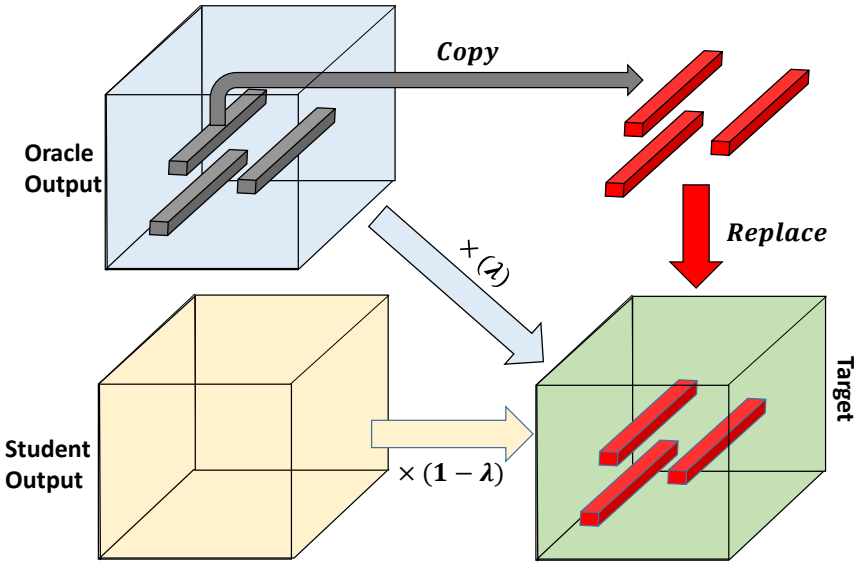

Figure 3.3: Target Tensor Composition.

To alleviate the downsides of both loss designs and still to preserve their advantages, I introduce a novel loss by a combination of them in Equation 5.7:

$$\begin{aligned} L_{final} &= \sum \|T_s^H - T_o^H\|_2^2 \\ &+ \sum \|T_s^E - ((\lambda * T_s^E) + ((1-\lambda) * T_o^E))\|_2^2, \end{aligned} \tag{3.3}$$

where $T_s^H \& T_o^H$ are the student and oracle cells with a high chance of object existences and $T_s^E \& T_o^E$ are the cells with a low expectation. More specifically, the first part on the left side of Eq. 5.7 calculates the MSE between the parts which have high confidence of objects. The second part calculates a modulated MSE between the cells with a low expectation from both the oracle output tensor and the student output tensor. Here, λ is the modulation factor. Figure 3.3 shows the procedure of creating the target tensor.

By using this loss function, the student model will have a lower chance to generate extra false positives. Also, it would not strictly force the student model to mimic the oracle exactly. I aim to partially address the challenges 1) and 4) in Sec. 3.3.1, with such a fast and effective loss function.

3.3.5 Key Frame Selection

Another crucial module to enable TKD working properly is a procedure to demonically select the time instances to train the student model during the inference stage. Specifically, TKD seeks the frames that by training over them the model has a higher expectation of reducing the loss, thus eventually improves the detection accuracy. For the rest of this document, I denote these frames as the key frames.

Selecting a larger number of frames as the key frames will hurt the performance since re-training is computationally expensive; While, selecting too few number of frames will hurt the detection accuracy as the student may not align well with the oracle model in time. Thus, an effective and fast procedure to select the key frames

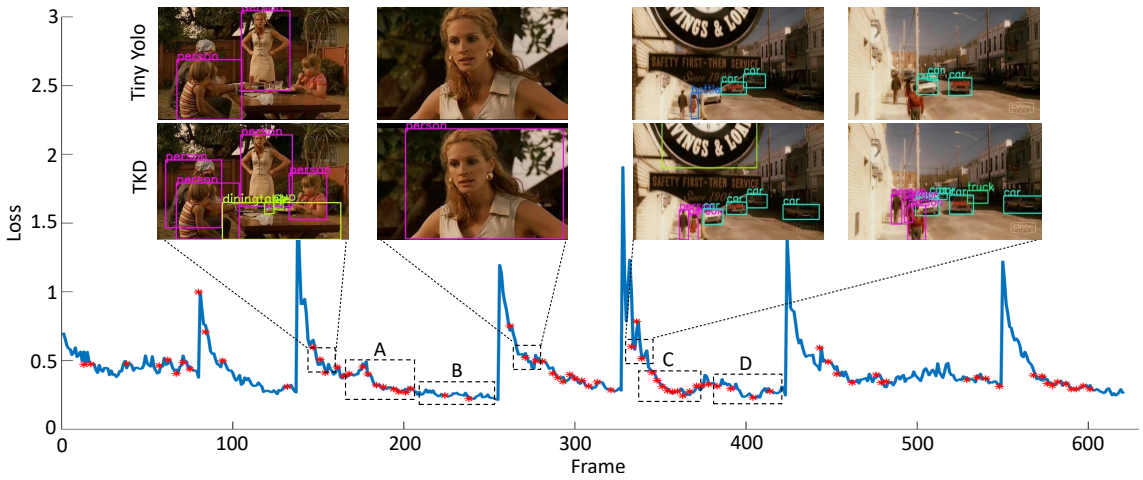

Figure 3.4: Key Frames Selected Using TKD over Two Scenes from the Hollywood Scene Dataset Marszałek *et al.* (2009). The Red Crosses Indicate the Key Frames Selected by My Method. See Further Discussion in Sec. 3.4.3.

is highly desired to yield a positive effect on the system's performance.

I propose a key frame selection procedure which is both efficient and also practical. First, I check the training prevention factor τ. If the student model has been trained in any last τ frames; I will exit the key selection procedure. It is based on the reasonable assumption that if we have any environment change, it typically takes τ frames that this change be fully observable. Thus, when I train the student, training for the next τ frames would not be beneficiary. Second, I start my decision process which I formulate in Equations 5.11:

$$
\begin{aligned}
I \in \{0,1\} &\begin{cases} 0 & \textit{Do not distill knowledge}, \\ 1 & \textit{Distill knowledge}, \end{cases} \\
I &= LSTM(F_s) \ \vee \ I_R, \quad I_R \sim B(2, P_t), \\
P_t &= \begin{cases} max((P_{t-1} - 0.05), 0.05) & \Delta L < \sigma, \\ min(2P_{t-1}, 1.0) & \Delta L > \sigma, \end{cases}
\end{aligned}
\tag{3.4}
$$

where I is the indicator that denotes our final decision. It takes the disjunction of the LSTM's output and the random module's output. I pass the features extracted

from the student model F_s (the last layer before the decoder) to the LSTM module (with one LSTM layer & one fully connected layer) which outputs a signal indicating to train the student model or not. Here, it is worth to note that I introduce another binary random module I_R (with binomial distribution $B(2, p_t)$) which decides in a random fashion to train the student model or not. The random procedure is added as a safeguard in case the LSTM model outputs a sequence of erroneous decisions. At the end, I update the LSTM module based on the result feeding back after the training procedure. If the LSTM makes a correct decision where the observed loss decrease $\Delta L < \sigma$ where in my experiments $\sigma = -0.05$, the random factor P_t will be reduced by 0.05. If the LSTM model makes a wrong decision, I update the LSTM model and double the random factor P_t. Figure 3.4 shows an example output of key frames selected by my method. I apply knowledge distillation selectively to a few number of frames which partially addresses the aforementioned challenges 1) and 2) in Sec. 3.3.1.

3.4 Experiments

The presented theoretical framework suggests three hypotheses that deserve empirical tests: 1) TKD can perform visual recognition efficiently, without hurting the recognition performance significantly; 2) the novel loss function can improve online training of the decoder; and 3) with the TKD frame selector mechanism, the overall system yields the best performance over other key-frame selection mechanisms, by locating the key frames more accurately (frames which training over them can improve TKD accuracy).

To validate these three hypotheses, I evaluate TKD on the Hollywood scene dataset Marszałek *et al.* (2009), YouTube-Objects dataset Prest *et al.* (2012), The Pursuit of HappynessMuccino (2008) and the office Daniels (2013). I have trained all

the base models (RetinaNet Lin *et al.* (2018), FasterRCNNRen *et al.* (2015), Yolo-v3 and Tiny-Yolo Redmon and Farhadi (2018)) over MS COCO dataset Lin *et al.* (2014). I implemented the TKD as described in Sec. 3.3.2 with two different configurations. First, I perform the process of inference and distillation sequentially among the same thread; the other way, I perform the distillation in a separate thread and run the student and oracle in parallel, both architecture implemented using the PyTorch environment Paszke *et al.* (2017). All experiments are carried out on one single NVIDIA TITAN X Pascal graphics card.

Hollywood scene dataset Marszałek *et al.* (2009) has 10 classes of scenes distributed over 1152 video. In this dataset, videos are collected from 69 movies. The length of these video clips are from 5 seconds to 180 seconds. Length and diversity of video clips make this dataset a perfect candidate to evaluate the key selector method and the novel loss function.

YouTube-Objects dataset Prest *et al.* (2012) is a weakly annotated dataset from YouTube videos, 10 object classes of the PASCAL VOC ChallengeEveringham *et al.* (2010) has been used in this dataset. It contains 9 and 24 video clips for each object class which length of these videos are between 30 seconds to 3 minutes. I used this dataset to evaluate TKD overall performance due to its high-quality objects level annotations.

The pursuit of happyness Muccino (2008) & The office Daniels (2013) are two famous movie and TV series. This two video clips contains several scenes which have the smooth transitions. The Pursuit of happyness serves a great testbed since it has scenes in different locations such as office, street, etc. It is also more close to the real world scenario from a camera of the intelligent agent. Also, the Office is selected as most of scenes have been recorded in the same location which make it suitable for testing the novel loss function.

Method	Hollywood Scene Dataset						The pursuit of happiness					
	IOU=0.5		IOU=0.6		IOU=0.75		IOU=0.5		IOU=0.6		IOU=0.75	
	AP	F-1	AP	F-1	AP	F-1	AP	F-1	AP	F-1	AP	F-1
Random Selection	0.71	0.75	0.54	0.68	0.48	0.49	0.65	0.65	0.55	0.58	0.35	0.43
Scene Change Detection	0.68	0.58	0.47	0.50	0.23	0.35	0.54	0.58	0.45	0.53	0.35	0.44
Tiny-Yolo	0.45	0.16	0.38	0.14	0.10	0.28	0.37	0.11	0.25	0.10	0.08	0.06
Tiny-Yolo (73%) + Yolo-v3 (27%)	0.60	0.49	0.59	0.49	0.44	0.46	0.58	0.47	0.52	0.46	0.39	0.44
TKD	**0.75**	**0.76**	**0.58**	**0.69**	**0.49**	**0.50**	**0.73**	**0.67**	**0.59**	**0.61**	**0.40**	**0.46**

Table 3.1: Performance of TKD with Different Training Methods over Hollywood Scene Dataset and the Pursuit of Happiness.

3.4.1 Ablation Study

As shown in table 3.1, I compare different strategies to highlight the effectiveness of my proposed novel loss and key frame selector. I consider the output of oracle model as ground truth and evaluating different methods over it. Here, I compare five methods: 1) TKD with random key frame selection; 2) TKD with Scene Change detection; 3) Tiny-Yolo without any training; 4) Combination of Tiny-Yolo and Yolo-v3 without training; and 5) TKD with my proposed key frame selection method.

In the following experiments I have set the λ to be 0.4 which is obtained heuristically. In 3.4.3, I will go through the findings which I observed in my search for the best λ.

Random Selection: Here, instead of selecting key frames by my proposed method, decision modules selects frames purely randomly for further processing. During the testing phase, the probability is set to be 27% (to make sure it selects more frames than my method (25% in average)). Random selection achieves 0.75 F_1 score (IOU=0.5) in Hollywood scene dataset and achieves 0.65 F_1 score (IOU=0.5) in the pursuit of happiness. On average, it reaches a frame-rate of 89 frame per second (FPS).

Scene Change Detection: This method uses the content-aware scene detection

method Castellano (2018). It finds areas where the difference between two subsequent frames exceeds the threshold value and used them as key frames for training the student. I selected the threshold with the highest performance and accuracy to report. This method achieves 0.58 F_1 score and 0.58 F_1 score in Hollywood scene dataset and The pursuit of happyness respectively. This method selected 24% frames as key frames ultimately. On average, the system yields a 93 FPS.

Tiny-Yolo without any training: I test Tiny-Yolo Redmon and Farhadi (2018) to show the accuracy of a strong baseline model without temporal knowledge distillation. This model achieves 0.16 F_1 score and 0.11 F_1 score in Hollywood scene dataset and The pursuit of happyness respectively, which are significantly lower than other mentioned methods. However, This model has 220 FPS, the fastest among all.

Tiny-Yolo + Yolo-v3 without training: In this configuration, I used Tiny-Yolo and Yolo-v3 v3 Redmon and Farhadi (2018) together. I designed a random procedure which runs Yolo-v3 with a probability of 27% and Tiny-Yolo for the rest of times. This model achieves 0.49 F_1 score and 0.47 F_1 score in Hollywood scene dataset and the pursuit of happyness respectively. Frame-rate approaches 89 FPS.

TKD with the key frame selection method: Initially, I set τ (the training prevention factor) to 2 (I observe that the transition between two scene takes at least 2 frames); along with setting the minimum random selection to 5%. In Hollywood dataset, my method selects around 26% of frames and the F_1 score achieves 0.76 (IOU=0.5). In The pursuit of happyness movies, my method selects around 24% of frames and the F_1 score reaches to 0.67 (IOU=0.5). In average, the system achieves a frame-rate of 91 FPS sequentially and 220 FPS with running inference and knowledge distillation in parallel.

Table 3.1 lists the experimental results I observed with these variants. These experiments shows, the TKD, while maintaining a similar frame-rate as other methods,

Method	IOU=0.5	
	mAP	F-1 score
RetinaNet-50Lin *et al.* (2018)	0.45	0.44
FasterRCNNRen *et al.* (2015)	0.52	0.50
Tiny-Yolo Redmon and Farhadi (2018)	0.38	0.33
Tiny-Yolo (73%) + Yolo-v3 (27%)	0.44	0.45
TKD	**0.56**	**0.55**
Oracle (Teacher)		
Yolo-v3Redmon and Farhadi (2018)	0.60	0.62

Table 3.2: Compression of Accuracy (Iou=0.5) over Youtube Object Dataset.

it can achieve higher recognition accuracy. To further validate this claim, I conduct one additional experiment on a single-shot movie Mokri (2013), TKD selects 21% and random procedure selects 27% of the total frames for re-training. They reach comparable F1-score (TKD:0.807, Random:0.812), but my TKD method uses 10400 frames less than the random one.

3.4.2 Overall Performance

Table 3.2 shows mean average precision (mAP) and F_1 score for five different object detection models as well as my TKD method over the Youtube object dataset Prest *et al.* (2012). For the student models without oracle's supervision, I train them to the best performance I could achieve. Not surprisingly, larger or deeper models with larger numbers of parameters perform better than shallower models, while smaller models run faster than larger ones. However, TKD achieves a high detection accuracy compare to RetinaNet, FasterRCNN, Tiny-Yolo, combination of Tiny-Yolo and Yolo-v3 (same configuration which is described in Sec. 3.4.1). TKD's detection performance also approaches the performance of the oracle model (Yolo-v3). In this experiment, 25% of frames has been selected for training using the proposed key frames selection method.

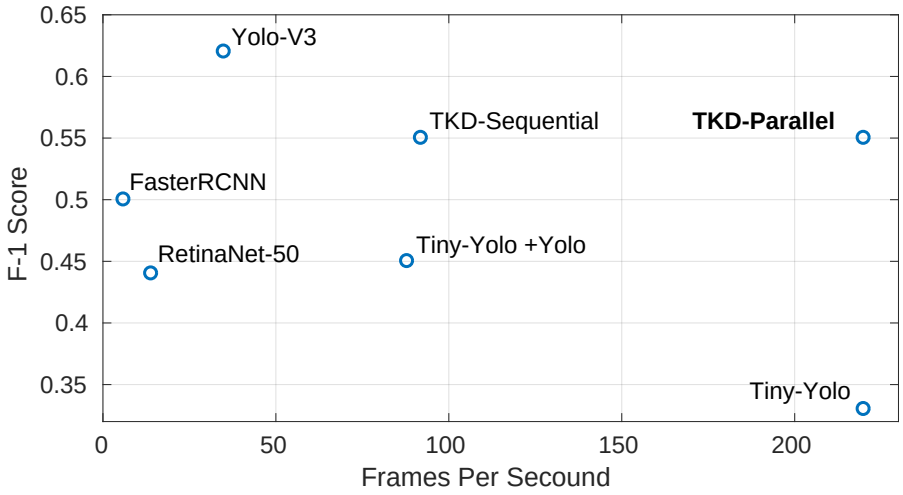

Figure 3.5: Accuracy and Speed in Youtube-objects Dataset.

To illustrate the accuracy-speed trade-off, I further plot them in Figure 3.5, where we can see that the TKD archives higher accuracy compare to other shallow methods while still operating far above the real-time speeds with a 91 FPS. The oracle model has a better detection accuracy, but it runs much slower than the TKD.

3.4.3 Further Study and Discussions

In this section, I provide further insight into the loss function design, the general knowledge distillation idea, and suggest an application of the proposed method.

Loss function: I studied λ effect over number of true positives and false positive generated by TKD. All tests are done over an episode from The office Daniels (2013). I choose this video since it was recorded in one indoor environment, with a consistent objects distribution. Table 3.3 shows the student model's detection accuracy varies with the different choices of λ. At $\lambda = 0$, I observed lower number of false positives since fewer number of frames (5%) selected by the key frame selection module. With a low λ (except at 0), I observe an increase in false positives as the model tries to generate more boxes and loss function doesn't punish hardly enough onto the student

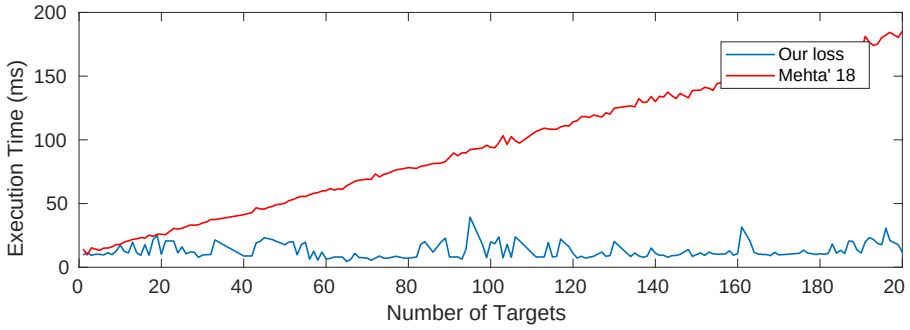

Figure 3.6: Computational Costs for Loss Functions.

ssd		**0**	**0.2**	**0.4**	**0.6**	**0.8**	**1**
IOU	**AP**	0.47	0.72	0.82	**0.83**	0.79	0.8
0.5	**F-1**	0.36	0.649	**0.676**	0.656	0.634	0.643
#TP		3353	8570	8371	7806	7274	7438
#FP		215	2952	1522	1129	814	841

Table 3.3: Parameter Study of λ over the TKD.

model for generating false positives. With a high λ I observe drops in the true positive rates since we are forcing the student to learn noises which are likely introduced by the oracle model. Consequently, 0.4 is empirically the best choice here, and I set it as the λ value for all the experiments.

To validate the loss design, I further compare its performance with the one from Mehta and Ozturk (2018). Mehta and Ozturk (2018)'s loss is based on Non-Maximum Suppression (NMS) algorithm running on the CPU which is computationally more expensive in comparison with my approach. Figure 3.6 depicts that, an increasing number of targets from each frame will result in the increasing of execution time for calculating the loss function for both methods. However, while achieving comparable mAP, my loss design has an almost constant execution time, while Mehta and Ozturk (2018)'s is linearly growing.

Temporal knowledge distillation: Here, I take a closer look at the key selection module. Figure 3.4 shows its performance over two video clips from the Hollywood

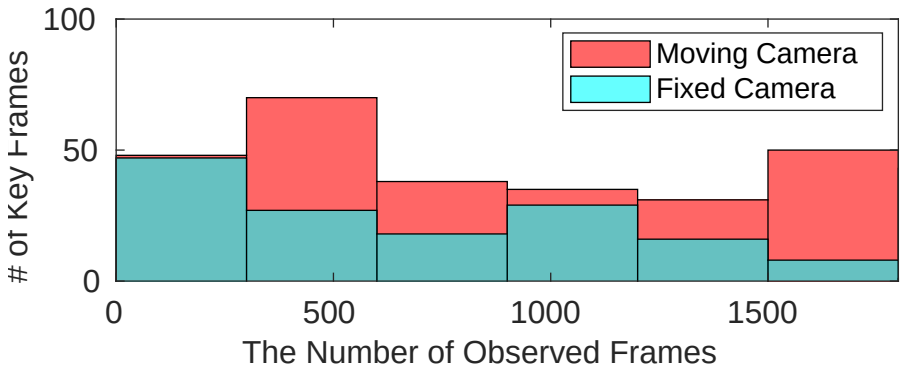

Figure 3.7: Key Frames Histogram.

scene dataset. Red crosses are frames which selected by my proposed method as key frames. At peaks, we have a scene change and logically these points would be the best candidate for training. Following this insight, I observe my model has a lag on detecting these points. Here, I argue that training over these frames are actually not the best ones for improving the student model's accuracy. Scene detection method is able to identify these points yet table 3.1 shows it achieves lower accuracy. Figure 3.4 shows the TKD after detecting a change in loss start stabilizing the model by selecting most of frames (parts A & C) and for the rest select less number of frames (parts B & D).

The proposed key frame selection method leads to improved performance comparing with Mullapudi *et al.* (2018)'s. Figure 3.7 shows that the number of selected key frames is adjusted based on the domain change. With the fixed camera case in which the domain does not change, the number of selected frames decreases along observing more frames (validated over UCF Crime datasetSultani *et al.* (2018)). Indeed, for the case of a moving camera, more key frames are selected to adjust the TKD to the specific domain. Here, the method presented in Mullapudi *et al.* (2018) relies on a static strategy of selecting frames which are chosen manually at the beginning.

For further evaluation, I applied TKD on one episode of the office TV series.

Then, I test the trained student model over another episode without any re-training at the inference time. I observed an increase of precision by 6% comparing to the case in which I use the original student model without applying TKD. The result demonstrates the domain adaption capability of my method. Furthermore, it maintains a high recall over other domains which indicates that unseen objects have a chance to be detected. With the method presented in Mullapudi *et al.* (2018), the model loses its generality over unseen objects due to the practice of optimizing the overall model with the new frames.

3.5 Conclusion

This chapter proposes a novel approach to distill temporal knowledge of an accurate but slow object detection model to a tinier model yielding a light and accurate object detection paradigm for robotic applications called TKD. I conducted experiments on the Hollywood scene dataset, Youtube object dataset, the pursuit of happiness movie, and the office TV series. I empirically validated that TKD maintains a high inference efficiency while achieving a high recognition accuracy. The accuracy even approaches the original oracle model for the object detection task.

The promising experimental results I observed suggest several potential lines of work which can further improve the system's performance. The frame selection procedure and training over the whole key frame are not efficient, and there are several concerns:

- A new object can be added to the environment, and the key frame selection does not detect this change on time.
- The key frame selector chooses unnecessary frames for retraining.
- The TKD passes the whole key frame to the oracle; however, the system has

uncertainty over a small region, not the entire frame.

In the next chapter, I studied the effect of mentioned concerns on system performance. I have also introduced a framework that can mitigate these concerns in an embedded environment with limited resources.

Chapter 4

TEMPORAL ADAPTATION ON EMBEDDED SYSTEMS

4.1 Introduction

The Internet of Things (IoT) refers to a world in which almost any *thing* is instrumented with sensors, computers and communication devices. These *embedded* systems will be utilized in a wide range of domains including surveillance, retail, healthcare, transportation, industrial robotics and many more. The emergence of IoT is taking place alongside a radical change in how the captured data is processed, namely, with the use of deep neural networks (DNNs). They have become the dominant algorithmic framework for extracting valuable information from massive amounts of disparate data for the purpose of prediction, classification and decision making.

DNNs are computationally intensive algorithms that involve several layers of nodes that perform billions of multiply-accumulate operations on very large dimensional data sets. Thus, DNNs have to be executed on high performance, large capacity cloud servers. Unfortunately, this means that the massive amounts of data generated by the IoT devices need to be transferred to the cloud, which will soon make this approach infeasible due to the limited bandwidth, the unacceptably large latency, and the potential for compromising the security.

The preferred solution is to have some or all the data processed by a *user-end* device, which is the first recipient of the data (e.g. a smartphone (Mao *et al.*, 2019), or a smart surveillance camera (Xiong *et al.*, 2019)). However, the limited computation and storage capabilities and/or energy capacity of user-end devices precludes them from executing complex DNN algorithms (Farhadi *et al.*, 2019; Han *et al.*, 2016b).

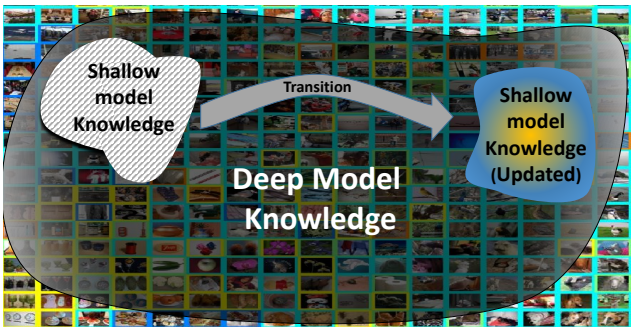

Figure 4.1: The Limited Knowledge of Shallow Model Can Be Adapted to the New Environment Using the Deep Model Knowledge.

Edge computing is aimed at addressing this problem by having a part or all of the computation performed on a more powerful local computer called an *edge* device that is connected to the user-end device via a local area network (LAN) or Wi-Fi connection.

In this chapter, I present a new approach to improve the execution of DNN algorithms in an edge computing environment. The proposed system targets complex DNN algorithms (e.g., RetinaNet (Lin *et al.*, 2017), Faster-RCNN (Ren *et al.*, 2015)) designed for *object detection* in digital images and videos in different domains (Figure 4.1). Object detection arises in practically every computer vision task and now plays a central role in nearly every one of the applications domains mentioned above. The goal of this work is to demonstrate how two devices – a lower performance user-end device and a much higher performance edge device can cooperate in the execution of computationally intensive DNN algorithms to achieve substantial improvement in energy consumption of the user-end device while achieving nearly the same quality of results as would be if the algorithms were executed solely on the more powerful device.

The proposed approach is based on a two-level hierarchy of models – a *Shallow* neural network (SHNN) (the *student*) that runs on the user-end device, and a DNN

(the *oracle*) that runs on the more powerful edge device. The edge device can also execute the SHNN. The use of the SHNN exploits an important characteristic of images, namely, that in any given image over a period of time, the diversity of objects is quite limited (Farhadi and Yang, 2019), and therefore, a DNN may not be necessary and a smaller, shallower model will suffice. On the other hand, when changes do occur, they must be detected, and the shallow model must be updated.

As shown in Figure 4.1, my approach detects such changes, activates the DNN as required, which in turn transfers the new knowledge (the encoding of the ground truth in the new weights) from the edge device to the user-end device to update the shallow model. All of this is done while performing inference, i.e., at run-time. The knowledge transferred includes the weights in the decoder layers (layers that detect the objects using the extracted features from previous layers). This transfer enables us to improve the inference time and energy consumption while having a tolerable accuracy loss compared to the deep model.

I demonstrate these ideas by implementing the proposed approach on a pair of devices where the user-end device is an NVIDIA Jetson Nano development kit and the edge device is a Dell workstation with NVIDIA Titan Xp GPU. The experiments show that the proposed method can achieve the desired accuracy with significantly lower inference time. Moreover, the total energy consumption of the user-end device was reduced by **78%** when compared to running the DNN entirely on the user-end device. Moreover, the results show that the ratio of object detection accuracy to the energy consumption is improved significantly using the proposed approach.

4.2 Background and Related Work

In this section, the background on the object detection methods and the metrics for evaluating their accuracy are described. Different categories of related work are

described and the drawbacks of each group are discussed.

4.2.1 Background

Object detection methods: Existing methods for object detection using CNNs can be classified as either two-stage or one-stage approaches. In two-stage methods such as FasterRCNN (Ren *et al.*, 2015), R-FCN (Dai *et al.*, 2016), and AdaScale (Chin *et al.*, 2019) classification and localization are implemented using two separate steps involving classification and region proposal. The one-stage approaches (such as Yolo (Redmon and Farhadi, 2018), SSD (Liu *et al.*, 2016), and RetinaNet (Lin *et al.*, 2017)) classify and localize objects in one step. One-stage detection models are generally faster while the accuracy of two-stage models is higher. However, at a smaller intersection of the ground-truth and the predicted object (intersection of union = 0.5), one-stage models can achieve nearly the same accuracy of the two-stage methods. In this chapter, I use single stage models, since they are better suited for embedded devices with limited computation resources.

Metrics for evaluation of detection accuracy: There are three main validation metrics in object detection: Recall, Precision, and F1 score (Powers, 2011).

Recall: This is the number of correctly detected objects divided by the total number of objects in the scene. Recall is crucial in safety-critical systems where missing an object in the scene could be catastrophic.

Precision: This is the total number of correctly detected objects divided by the total number of detected objects. This is useful for evaluating the systematic errors of detection.

F1 score: This is a measure of overall detection accuracy and is defined as $2 \times \frac{Precision \times Recall}{Precision + Recall}$.

4.2.2 Related Work

Implementation of object detection on cloud

Neurosurgeon (Kang *et al.*, 2017) and JointDNN (Eshratifar *et al.*, 2018) are two recent examples of performing image classification collaboratively between an user-end device and a cloud server. However, they do not consider object detection methods, which are a superset of image classification methods. *Glimpse* (Chen *et al.*, 2015b) performs object detection on mobile devices using a cloud server. When the network delay exceeds a certain threshold, their approach uses tracking to estimate the location of objects based on an active cache of frames.

MobiEye (Mao *et al.*, 2019) is another cloud-based object detection system for mobile devices implemented in a multi-threaded asynchronous manner. The first thread sends the key frames to the cloud for object detection. The second thread performs the object tracking based on the result of processed key frames using optical flow network. The response time of object detection for key frames is dependent on the network delay. Therefore, when the network delay is high, the newly observed objects in the scene may be missed.

Domain adaptation

Domain adaptation refers to training a model for a specific domain of observation. There is a substantial body of work on domain adaptation in object classification and detection methods (Lu *et al.*, 2017; Chen *et al.*, 2018; Dai and Van Gool, 2018). These methods have been presented to deal with challenges such as low quality images, and large variance in the background. This variation can result in a domain change in the training, validation, and test sets. However, existing methods do not consider sudden changes in the scene when performing inference at run-time.

Knowledge transfer

Knowledge transfer (a.k.a knowledge distillation) (Mullapudi *et al.*, 2019; Farhadi and Yang, 2019) has been widely used to improve the accuracy of object detection systems. This method transfers the knowledge of a deeper model (oracle) to a lighter model (student) with fewer parameters. If the shallower model can gain this knowledge, it will have the same accuracy as the deeper model while using fewer resources. However, the student model would not adapt to this knowledge since it has fewer parameters but it can adapt partially.

Mullapudi *et al.* (2019) proposed an image segmentation method where the shallow model is trained periodically at fixed intervals. The selection of fixed interval may not be efficient. In chapter 3, I (Farhadi and Yang, 2019) have proposed a systematic procedure using Long Short-Term Memory (LSTM) to determine the intervals at which the training of shallow model should be done. However, running the LSTM would be expensive on resource-confined user-end devices.

In this chapter, a knowledge transfer framework is proposed in edge computing environment for energy-constrained user-end devices. This framework is explored in different ways using real-world direct measurements.

4.3 System Framework

4.3.1 Knowledge Transfer

In this section, the idea of online knowledge transfer in the edge computing domain is described. Moreover, an efficient method is proposed to select the frames that should be sent to the edge device for re-training the shallow model.

In most vision datasets, there are a variety of domains that the model needs to learn. However, in real-world applications such as surveillance cameras, we are con-

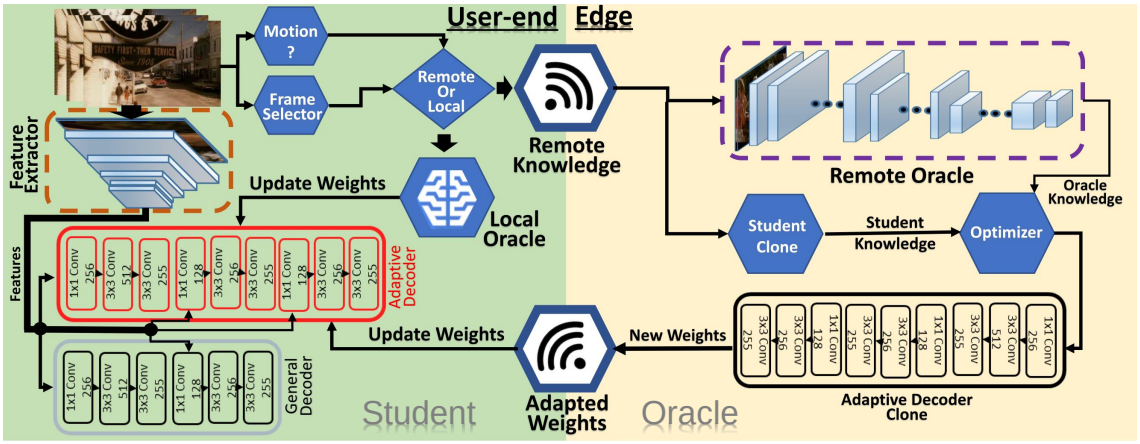

Figure 4.2: An Overview of Knowledge Transfer Method: The Main Thread of Execution on the User-end Device Runs a Shallow Student Model to Detect Objects. To Keep the Desired Accuracy, a Key Frame Selection Module Decides to Retrain the Student Model Based on the Oracle Model.

fronted with a specific domain and with limited types of objects. Although the scene can change, the changes are typically slow. Here, the oracle knowledge over this temporal domain can be used to adapt the student model to the observing environment (which has been called temporal knowledge distillation (Mullapudi *et al.*, 2019)). This approach can improve the accuracy and response time.

Although methods of knowledge transfer can improve the performance significantly, their implementation poses several challenges: 1) models may lose their generality and may not be able to detect objects seen for the first time; 2) training the student model for each and every incoming frame incurs high computation cost; 3) training a student model on the embedded device can affect other simultaneously running tasks, whereas training over the network incurs delay for adaptation. I address these challenges in the proposed approach.

4.3.2 Main Architecture

My method of knowledge transfer between a shallow and deep model extends the approaches described in (Farhadi and Yang, 2019; Mullapudi *et al.*, 2019) to energy-constrained embedded systems. Figure 4.2 shows an overview of the proposed architecture which consists of three main parts: 1) the student model which is a shallow CNN model; 2) the oracle model which has a deep structure that can reach the state-of-the-art accuracy in detection; and 3) the key frame selection method which selects the epochs at which student adaptation occurs.

The user-end device observes a scene and detects objects in the scene. Meanwhile, it adapts itself to the observing environment to improve the overall accuracy. In the following, each of these modules is described. Note that the presence of the oracle model in the user-end device is for evaluation purposes in the approach.

Student

This is a shallow model with a limited number of parameters requiring much less computation than a deep model. The student can learn only a limited amount of knowledge due to fewer parameters. Here, the student has a similar structure to Tiny-Yolo v3 (Redmon and Farhadi, 2018) with some modification to the decoder part. The student model (see Figure 4.2) consists of three parts: 1) a feature extractor, 2) a general decoder, and 3) an adaptive decoder. The feature extractor, which has the same structure as the base model (Tiny-Yolo) is trained on a conventional object detection dataset. The general decoder (same structure as base model) detects objects using the extracted features and is trained along with a feature extractor during the training stage. Finally, the adaptive decoder is optimized during inference time using the oracle knowledge. Another stack of convolutions has been added to the adaptive

decoder to improve the detection accuracy of small objects (Figure 4.2).

Oracle

This is a model with a larger number of parameters and a deeper structure and as a result can reach the state-of-the-art accuracy over the target dataset (called oracle model). This model will be used to extract knowledge over the temporal domain. The knowledge will be used for adapting the student model to the observing environment. Here, Yolo v3 is used as the oracle model due to the similarity of structure to that of Tiny-Yolo and its lower latency compared with two-stage object detection methods. This makes the optimization procedure more efficient at inference time.

Optimizer (weight update)

To transfer knowledge from the oracle to the student model, *Adam* gradient descent method (Kingma and Ba, 2014) is used. First, the distance between the student and oracle model needs to be calculated. The student and oracle both have three output matrices with the same size. By calculating the $L2$ distance of student and oracle outputs ($\sum \|T_i^S - T_i^O\|_2^2 \, , i = [1, 2, 3]$), the distance of two models is calculated. Next, using this distance as the loss value, the optimizer updates the weights of adaptive decoder in the student model. After adaptation, the weights of running student model will be replaced by the new weights.

Key frame selection

It is important to know when we need more knowledge while observing the environment. Training over all incoming frames will be expensive while training over a number of frames is needed to maintain accuracy. Mullapudi *et al.* (2019) proposed a static interval key frame selection that is selected by the user at the beginning. To

reach the desired accuracy, the interval should be small which will increase the inference time, energy consumption, and hardware utilization. In chapter 3, I (Farhadi and Yang, 2019) proposed a combination of a uniform selector and an LSTM module to select key frames. This method achieves higher performance compared to Mullapudi *et al.* (2019). However, running the proposed LSTM module on a resource-constrained device would be costly.

In the proposed approach, Kalman filtering (Kalman, 1960) is used to track changes on the scene. If there is a significant change on the scene compared to the last adaptation, the frame is a good candidate for re-training. Also, frames are selected using a binomial selector described in equation 5.11:

$$
\begin{aligned}
&I \in \{FALSE, \quad TRUE\} \\
&I = Motion(F_S, F_L) \ \wedge \ I_R, \quad I_R \sim B(2, P_t), \\
&P_t = \begin{cases} max((P_{t-1} - 0.05), 0.05) & \Delta L < \sigma, \\ min(2P_{t-1}, 1.0) & \Delta L > \sigma, \end{cases} \\
&F_S : Observed \ frame, F_L : Last \ key \ frame,
\end{aligned}
\tag{4.1}
$$

where I denotes the final decision of the selector indicating whether or not to re-train. The I_R is sampled from the binomial distribution. The probability of selection (P_t) is changed based on knowledge transfer loss. If ΔL (i.e. the difference between current loss and previous training loss), improves more than σ, the probability factor will be doubled to increase frames for knowledge transfer. Otherwise, P_t will be decreased to select fewer frames for training. At least 5% of frames are selected as a sample to avoid missing the scene changes. The value of σ is a hyper parameter which is obtained after several rounds of experiments. This value can be changed based on the type of loss function. Here, the value of σ is chosen to be $\sigma = 0.5$.

Moreover, we will not select any other frame for adaptation while we are processing

a key frame. This approach makes the detection system more adaptable to the newest observed objects by avoiding the queuing of frames.

Knowledge source

The student module can be updated using local or remote knowledge. As shown in Figure 4.2, the detection system has its own Oracle (local Oracle) to update the weights and adapt to the environment. The local Oracle is both optimizer and oracle model. Also, the user-end device can request a network oracle (edge) for a deeper knowledge and optimization. The edge has a clone of student model and updates this clone and sends it back to the user-end device. This approach is not on-time due to network latency. Section 4.4 includes a detailed evaluation of these trade-offs.

Multi-threading

To minimize the inter-effect of these sub-modules, each module is running on a separate thread. Although these modules are running independently, they may affect each other due to limited resources on the device. For instance, the oracle module can use most of the available CUDA cores while we are running the student model and cause an increase in inference time.

4.4 Experimental Results

In this section, the experimental setup and the results of applying the approach are described.

4.4.1 Setup

Hardware Setup: The user-end device is NVIDIA Jetson Nano developer kit (jet, 2019) which is the first recipient of camera frames. This kit is equipped with a

quad-core ARM A57 CPU operating at 1.43 GHz and a 4GB 64-bit LPDDR4 RAM. Moreover, it has a 128-core Maxwell GPU. The edge device is a Dell workstation with an Intel Xeon W-2125 CPU operating at 4.0 GHz having 32GB of RAM and an NVIDIA Titan Xp GPU.

Both Local Area Connection (LAN) and Wi-Fi were tested as the communication medium between devices (tested in a network with a WAN (wide-area network) backbone). The bandwidth of Wi-Fi and LAN connections were approximately 13 and 100 Mb/s respectively. To measure the power consumption of the user-end device, Monsoon HV power monitor was used (Figure 4.3).

Dataset: To verify the efficiency of the proposed approach, two types of videos from fixed and moving cameras were selected. The fixed camera case was from the surveillance videos in the UCF dataset (Sultani *et al.*, 2018). The moving camera case was from the car crash dataset (Chan *et al.*, 2016). The initial weights of student and oracle model were obtained by training on the Microsoft COCO dataset (Lin *et al.*, 2014).

Evaluation Metrics: The proposed approach to train on the edge device over a network was evaluated in terms of accuracy and performance metrics. The F1 and Recall scores are representative metrics for the accuracy of the detection method.

In the experiments, the output of deep model was assumed to be the ground-

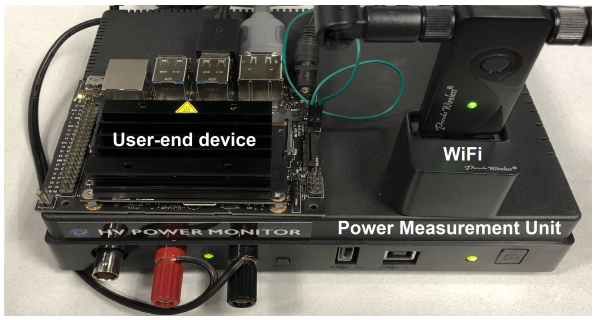

Figure 4.3: Experimental Setup.

Table 4.1: Comparison of Different Approaches. Local Training and Network Training Both Use the Proposed Key Frame Selection Method Using Full Precision Data. The Energy Column Is the Average Energy Consumption for Each Frame. Overall Score Is the Ratio of F1 Score to Energy.

	Metrics			
	Energy (J)	Inference Time (s)	F1 Score	Overall Score
Shallow Model	1.06	**0.187**	0.489	0.46
Deep Model	4.55	0.669	1	0.22
Local Training	1.83	0.215	**0.753**	0.41
Network Training (Wi-Fi)	1.25	0.198	0.731	0.58
Network Training (LAN)	**0.98**	**0.193**	0.745	**0.76**

truth and the proposed approaches were compared with the knowledge of deep model. Moreover, the average inference time and training time for all the processed frames was measured. Additionally, the total energy consumption of the user-end device processing the whole video was compared. This energy consumption is attributed to different factors including video decoding, inference, non-maximum suppression (NMS) post processing, and network communication. In order to evaluate the overall efficiency of the approach, both in terms of accuracy and energy consumption, the overall score is used as the ratio of F1 score to energy consumption (DAC, 2018; Alyamkin *et al.*, 2018).

Test Scenarios: The test scenarios include three cases:

1. Network Training (NT): the weights are updated on the edge device through either Wi-Fi or LAN connections. The input data is a tensor of size $416 \times 416 \times 3$ which needs to be transmitted from the user-end device to the edge device. Moreover, the weights of decoder (explained in Figure 4.2) will be sent from the edge to the user-end device.

2. Local Training (LT): The weights are updated locally on the user-end device.

In this approach, the calculation of loss function to update the weights is also done on the user-end device.

3. No Training: The weights are not updated at all and the inference is done using the shallow model.

The aforementioned scenarios were tested using full precision and half precision weight and data values on the videos of both fixed and moving cameras. The efficiency of key frame selection (KFS) method was compared with the case in which all the frames were selected as the key frame (w/o KFS).

4.4.2 Experimental Results

Table I shows the comparison of LT and NT versus running the deep or shallow model on the user-end device for the fixed camera video. In case of NT using LAN, the energy consumption and inference time were reduced by around 78% and 71% compared with running the whole deep model on the user-end device while the F1 score was reduced only by $\approx 25\%$. However, as shown in (Farhadi and Yang, 2019), the accuracy of student model using knowledge transfer remains almost the same as the deep model in Youtube video dataset (Prest *et al.*, 2012). On the other hand, running the shallow model on the user-end device resulted in unacceptable accuracy (F1 score = 0.489). Note that training the shallow model locally led to additional 87% energy consumption. The overall efficiency of the approaches was evaluated using the overall score (i.e. the ratio of F1 score to energy). The overall score of network training (LAN) leads to 1.65x and 3.45x improvement compared with running the shallow and deep model on the user-end device. The reason for getting better overall score in LAN compared with Wi-Fi is due to the lower communication delays in the network. Still, the network training using Wi-Fi can gain better overall score

compared with other approaches.

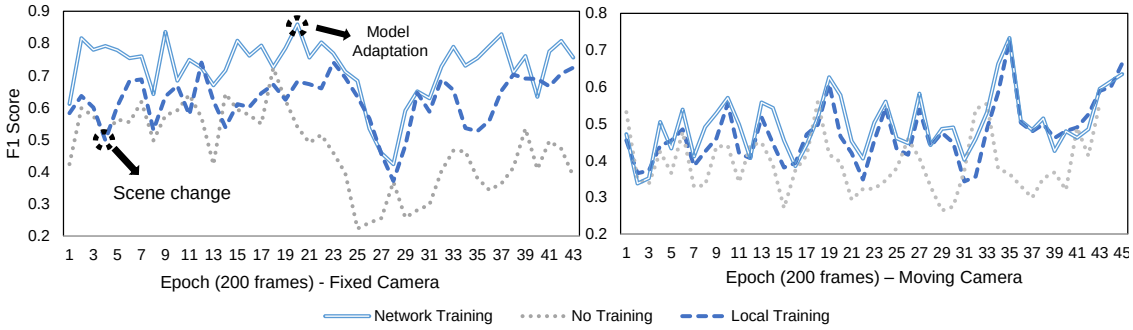

Figure 4.4: F1 Score Variation. In the Case of Fixed Camera, the Network Training (NT) Using Wi-fi Connection Even Has a Better F1 Score in Comparison with the Local Training (LT). Both NT and LT Operate on Half Precision Data. The High Spike Indicates That the Model Has Been Adapted to the Environment While the Low Spike Shows the Scene Change.

The variation of F1 score was also measured (shown in Figure 4.4). Both NT and LT perform significantly better than the case in which no incremental training was done (both fixed and moving camera videos). Moreover, higher F1 score was achieved in the case of fixed camera videos. It demonstrates that the adaptation is more effective in fixed camera videos due to fewer changes in the scene. In the case of fixed camera, NT outperforms LT since training latency is lower which leads to higher achievable accuracy.

The presence of low spikes in some parts of Figure 4.4 suggests that there is significant scene changes in the video at those epochs. On the other hand, the high spike indicates the epochs that the detection system was able to adapt to the environment. Due to different re-training latency values, a shift in the spikes among different scenarios can be observed.

Figure 4.5 (a) shows the average Recall for the different scenarios mentioned earlier. Recall in the case of LAN connection was better than Wi-Fi. This is due to the fact that the data transmission latency of LAN connection was lower than Wi-Fi and the student model was updated more frequently. On the other hand, the accuracy

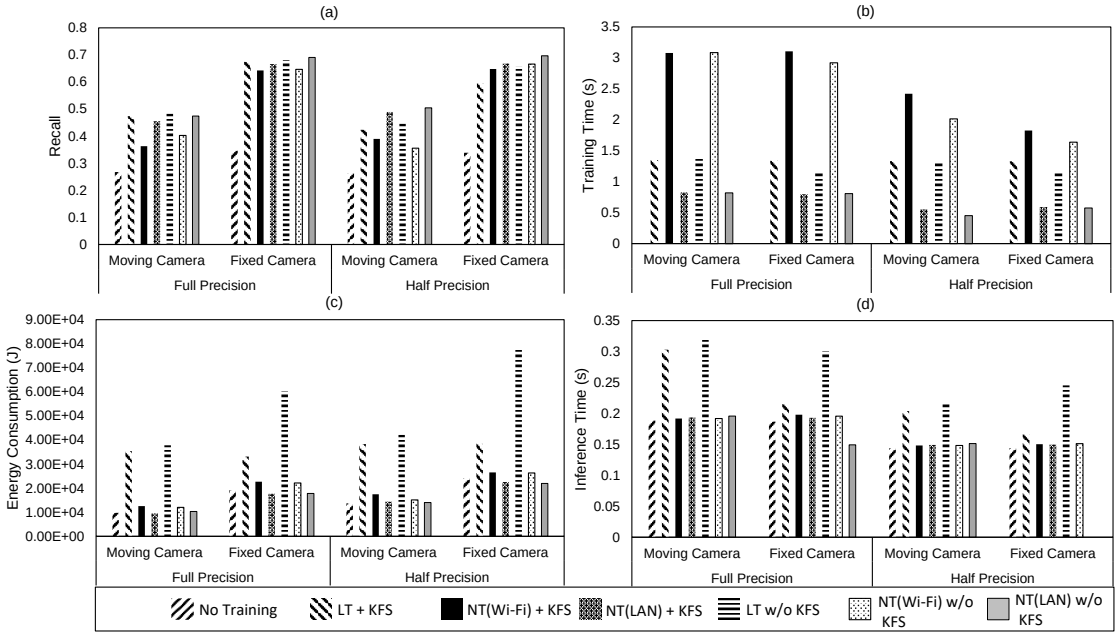

Figure 4.5: Comparison of (a) Recall, (B) Average Training Time, (C) Total Energy Consumption of All Frames, and (D) Average Inference Time, for Fixed and Moving Camera Videos Using Wi-Fi and LAN Connections. The Efficiency of Key Frame Selection Method (KFS) Has Been Also Compared with the Case in Which All the Frames Are Trained (w/o KFS).

of NT and LT are close even when using the low speed wireless connection since the training latency is the same. Moreover, training over the network using half precision data reduces the communication time for sending the data and weight values significantly while having a negligible effect on the accuracy. Note that the values of F1 score followed the same trend as with Recall.

Figure 4.5 (b) shows the average training time of all processed frames. Network training using a LAN connection has the lowest training time while training using the Wi-Fi has the highest among all cases. Using half precision data reduces the training time significantly specifically in NT using Wi-Fi connection. Even using half precision data, LT was still having lower training time than the NT (Wi-Fi) since the training time locally using half precision incurs lower computation.

The total energy consumption of the user-end device for different scenarios is compared in Figure 4.5 (c). The lowest energy consumption was achieved using NT with LAN connection. This is due to the fact that the transfer of weights to the edge device was done faster and the user-end device was not involved in the training procedure. Although the training time for NT (Wi-Fi) was highest, the energy consumption is close to NT(LAN) case (NT (LAN) achieves higher accuracy as mentioned before). It can be also seen that LT leads to higher energy consumption compared with NT. There are two reasons for this observation: 1) The high computation cost of running the oracle model locally; 2) Increase of inference time due to interference of local training with the online inference.

Figure 4.5 (d) shows the average inference time of all frames. The effect of local training can be again observed in the higher inference time compared with the network training. NT(Wi-Fi) leads to lower inference time compared with NT(LAN). The reason is due to higher accuracy obtained in NT(LAN) scenario. Note that the post processing on the detected objects (NMS) took longer time for the scenarios with higher accuracy which led to higher inference time.

The efficiency of key frame selection method was also evaluated in the experiments. It can be observed in Figure 4.5 (a) that NT (LAN)+KFS performs closely in terms of Recall metric compared with the case where the training happens at all video frames (NT (LAN) w/o KFS). Moreover, the energy consumption and the inference time in the case of LT w/o KFS is significantly higher in comparison with LT+KFS since the re-training should happen for all the frames in LT w/o KFS.

4.4.3 Further Discussion

The experimental results gave us some insights on how to design the system for the implementation of online knowledge transfer. The takeaways can be summarized

as below:

- Local training: The energy consumption of this method is higher compared to other approaches. On the other hand, the training time is more predictable in comparison with network training using Wi-Fi. This is due to the fact that the communication time using Wi-Fi follows a stochastic behavior.

- Network training: This approach can lead to higher accuracy/energy ratio on average. Network training using LAN connection is as predictable as local training since the LAN connection is more stable compared with Wi-Fi.

- Loss Function: The used loss function in this section is based on the Euclidean distance of student and oracle model. However, the calculation of this loss function is computationally intensive on the CPU of user-end device. Therefore, local training can be more expensive in terms of energy consumption and inference time.

- Frame selection: The frame selection strategy selects more frames to train at the beginning of the environment observation. The number of selected frames is decreased throughout the observation of the environment. This observation was more vivid in fixed camera videos.

- Potential solution: Based on these observations, a combination of network and local training is suggested where the initial frame training can be done on the edge device. For the rest of the frames, a decision making policy can determine whether the training should be done locally or on the edge device. The decision making policy can be based on the stability of the communication media. Moreover, the number of frames to be trained affects the decision making policy. For instance, when the number of frames to be trained is higher, the network

training is a better option.

4.5 Conclusion

In conclusion, I introduced and implemented a framework for incremental knowledge transfer in an edge computing environment. The parameters of a shallow model running on the user-end device are updated during inference at some key frames to achieve the close accuracy as using a deep model. I demonstrated the proposed approach in the real-world scenario. The framework consisting of a shallow and a deep model resulted in 78% energy reduction when compared to running the deep model alone. The experiments also revealed that communication latency could substantially affect vision adaptation. Hence, we need another approach to transfer knowledge from oracle to student for real-time applications with high network latency. In chapter 5, I introduced a novel approach that, by focusing on the region of interests, reduces the knowledge transfer significantly.

Chapter 5

ARGOS: AN ADAPTIVE AND REGION-SCALE PROPOSER BASED OBJECT RECOGNITION SYSTEMS

5.1 Introduction

In recent years, there has been significant success in computer vision with applications such as object detection and instance segmentation. Although research in this field has been progressing rapidly, there is still a considerable gap between research and practical deployment. The vision-based intersection management (vIM) of CAVs is one of the emerging applications which will become an essential part of cities (Khayatian *et al.*, 2020). A study conducted by American Automobile Association (AAA) shows more than two people are killed every day in the U.S. due to accidents caused by red-light runners (Bomey, 2019). We face two main challenges in vIM: 1) The processing unit needs to be at the location; using cloud computing is not feasible as it requires an extensive network infrastructure that can support the required bandwidth for the cameras. Furthermore, network delays increase response time. 2) Existing DNNs are energy hungry, affecting deployment practicality for battery-powered or energy-harvested systems. Object recognition is the most energy and computational demanding module in vIM. This problem aggravated as vIM needs to be accurate and agile in an embedded environment with limited resources.

A new set of recognition models have been proposed to address the high computation cost of neural networks using new architectures (Iandola *et al.*, 2016; Sandler *et al.*, 2018; Tan *et al.*, 2019; Zhang *et al.*, 2018) or compressing models (Han *et al.*, 2015a; Wu *et al.*, 2016; Zhou *et al.*, 2016). These approaches consider all image

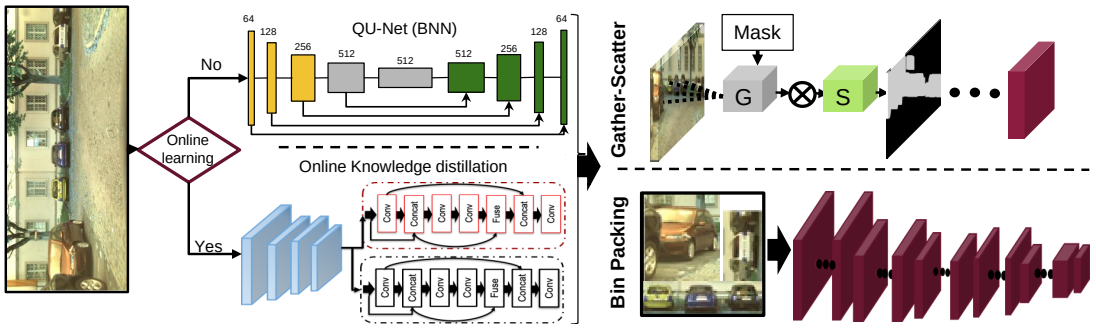

Figure 5.1: ARGOS Predicts the Regions of Interest in Images for Further Processing (Through Bin Packing or Gather-scatter)..

regions equally important and apply a model to all image pixels. Dynamic neural networks try to solve this issue by adopting gating mechanisms to control the depth of the model, such as Skipnet (Wang *et al.*, 2018b) and SACT (Figurnov *et al.*, 2017); or selecting regions that are important and process those independently, such as Dynamic Convolution (Verelst and Tuytelaars, 2020). Although these approaches showed promising results, their applicability is limited to residual neural networks (He *et al.*, 2016a).

In addition, new methods have been proposed for video object recognition which consider the temporal relationships between frames (Farhadi and Yang, 2020; Liu *et al.*, 2019; Mullapudi *et al.*, 2019) to reduce the computation cost and inference time. These models are inspired by the human visual system, which relies on contextual cues and memory to supplement their understanding of the environment (Oliva, 2005). These models use a light model for the inference time and adapt it to the environment using online knowledge distillation from a deeper model. Although this set of object recognition models can reduce the computation cost and thus the inference time, their predictive performance relies on deep features extraction on a few key frames. The key frame selection highly depends on how often significant scene change happens and the number of emerging new objects. Thus, the key frame mechanism becomes the

impeding factor preventing model deployment on embedded devices. Furthermore, the processing time of selecting and extracting features from the key frames can degrade system response time, switching from one scene to another as a deeper model is applied to the whole frame.

In this , I am considering two main characteristics in vIM application that help us meet the requirements with minimum resources: 1) cameras are fixed, and the domain of the observing environment will not change; hence we can use temporal knowledge adaptation and use a lighter model at inference time; 2) as cameras will be installed on traffic or street lights pole (15 feet, with 15-degree angle), the relative speed is significantly slower compared to other applications such as autonomous vehicles hence a high frame-rate detection is not needed. Moreover, target objects take a small portion of the whole frame. Hence, we can focus on location with a chance of an object's existence.

I focus on reducing the model computation complexity by reducing the model input size. Mainly, DNNs are applied to all pixels in visual feature extraction as there is a data dependency between pixels. Here, I use a light neural network to decompose images into sub-regions and apply a deep neural network to regions of interest (RoI). The light neural network is considered as a pre-processing step and can be implemented in various fashions.

First, I use a binary RoI proposal model, QU-Net (fig.5.1), that efficiently predicts regions with a high probability of object existence. QU-Net is a binary neural network (BNN) which generates a binary segmentation mask. The regions proposed by this module can have a single object or multiple objects (if objects partially occlude each other). As QU-Net uses a BNN backbone, its computation and memory cost is meager and can be applied to high-resolution images. Furthermore, ARGOS method is general, unlike Skipnet or Dynamic convolution, which is only applicable to residual

convolutional neural networks (CNNs).

Second, I extended this concept to methods based on online (or temporal) knowledge distillation. As mentioned, in these models, a light model adapts to the environment using a deep model output on key frames. An eminent issue is the key frame selection and processing using an expensive model. Even with scene changes only happening in parts of the input frame, models at the earlier stages of the pipeline have to process the key frame entirely. Following this observation, a straight-forward idea is to only process the RoIs and reuse previously extracted features for the rest to improve recognition execution performance at inference time.

I adopt the light model as RoIs proposal model (by reducing the network's confidence). I also use an event-based mechanism to improve RoIs identification. After decomposing the input frame to separate RoIs, I apply the deep model to those regions. Such a system design can reduce the computation by skipping processing a substantial portion of the input frame that are not challenging or have an object.

I provide two methods to process the RoIs in a neural network. First, the *gather-scatter* approach based on the implementation described in (Verelst and Tuytelaars, 2020). This method is more efficient than other sparse matrix operations because it gathers the elements into a single dense matrix before applying the convolution operation. However, this method requires the implementation of custom layers for each neural network. Bin-packing is another approach for processing the RoIs that extracts each rectangular region of interest from the image and packs them into a batch of frames. This method is highly flexible. It can be applied to any pre-implemented neural network with no layer-wise dependency, at the cost of adding extraneous areas.

To validate the effectiveness of ARGOS, I conducted extensive empirical studies on the COCO (Lin *et al.*, 2015), Cityscapes (Cordts *et al.*, 2016), WiseNet (Marroquin

Figure 5.2: ARGOS Experimental Set Up.

et al., 2019) and UA-DETRAC (Wen *et al.*, 2020) datasets. I compared ARGOS performance with Dynamic Convolutions (Verelst and Tuytelaars, 2020) as well as other state-of-the-art methods with a set of evaluation metrics, including recognition accuracy, processing time, and energy consumption. ARGOS tests on COCO and Cityscapes datasets showed a computation reduction of 25% and 57%, respectively, with a marginal average accuracy loss of 4% and 1.5%. Moreover, it reduces the computation by 20% and 40% compared to the dynamic convolutions (Verelst and Tuytelaars, 2020) approach in object detection and segmentation tasks while improving the mAP by 3% and 20%. It reduced the computational cost by 89% and 80% on UA-Detrac and Wisenet with a marginal decrease in accuracy, 3%. Figure 5.2 shows the prototype for vIM application.

5.2 Related Work

5.2.1 Dynamic Neural Networks

Dynamic neural networks (Han *et al.*, 2021b) are networks that can adapt their structures or parameters to varied inputs, allowing them to achieve superiority in terms of computational efficiency and accuracy. Recently, these networks formed a key part of several literature studies. Sample-wise dynamic networks take into con-

sideration that different inputs may have different computational demands. Skipnet (Wang *et al.*, 2018b), and SACT (Figurnov *et al.*, 2017) exploit this assumption and dynamically adjust the layers based on the complexity of the image. Temporal-wise dynamic networks rely on the fact that specific frames may contain redundant information. This information can allow the networks to execute the critical frames selectively. AdaFrame (Wu *et al.*, 2019b) is an example of this type. These methods rely on the assumption that certain input frames can be effortless to process while spatial-wise dynamic convolutions rely on region-wise complexity, which is more suited towards real-time scenarios since every image can contain challenging regions.

Spatial-wise dynamic convolutions exploit the fact that the different regions in an image contribute unequally to feature extraction and focus the execution on the regions with a probability of containing an object of interest. Background subtraction estimates the difference between successive frames to estimate the moving objects has been widely used in the realm of identifying the regions of interest for spatial processing by dynamic neural networks (Nguyen and Choi, 2020; Sengar and Mukhopadhyay, 2020). However, these methods work on certain fixed assumptions and lack the flexibility to handle natural movements in the environment as well as the movement of the camera. Pixel-level dynamic networks are a type of spatial-wise method that performs adaptive computation at the pixel level. Dynamic Convolutions (Verelst and Tuytelaars, 2020) is one such approach that uses a mask to execute specific regions with crucial information for feature extraction. Dynamic Convolutions rely on residual blocks, and after each block, create an attention mask and only process selected regions. Therefore, a smaller spatial area for processing help in reducing the overall computational cost of the network. ARGOS has the same functionality; however, it does not rely on features extracted from residual blocks, making it more general. Moreover, it can reduce the initial layer computation as we have the RoIs

using ARGOS pre-processing mechanism.

5.2.2 Quantized Neural Networks

In CNNs, the main contributors to the complexity of a model are the convolutional operations because of the significant overhead of the repeated multiplication functions. Many methods have been proposed to reduce this overhead. Reducing the bit-width of the activations and weights has been one of them that helps in reducing the complexity of the respective layers. Binarized Neural Networks (Courbariaux *et al.*, 2016), and XNOR Net (Rastegari *et al.*, 2016b) took it further by proposing methods to binarize the weights and activations of the entire network. However, an entire binarized neural network is not feasible because of the accuracy loss. A layer-wise priority for binarization is used to alleviate the problem (Wang *et al.*, 2018a). As the binarization of deep layers leads to a lower accuracy drop, they use a bottom to top approach.

To counter the issue of the accuracy loss during binarization, flexible models which supported the use of different bit widths were developed. Zhou *et al.* (2016) (DoReFa-Net) introduced a method to train models using low bit-width weights and activations as well as efficiently implement them on different hardware devices such as FPGA, CPU, etc. Furthermore, it provided an efficient method to quantize the different layers. But all these models proposed had been tested on image classification and object detection and still cannot be applied on real application due to high accuracy degradation. By considering this fact I am adapting BNNs as a pre-processing to improve the CNNs performance. The binary segmentation model can recognize the interest areas for various objects and reduce the computation of DNN by reducing the processing regions.

5.2.3 Online Knowledge Distillation

In chapter 3, I introduced an online Knowledge distillation mechanism which is an approach to transferring information from one model (teacher) to another (student) at inference time. This approach trains an efficient model to mimic the output of an expensive teacher (Farhadi and Yang, 2020), as a form of model compression. Early explorations of knowledge distillation focused on using the teacher's rich output to train the student over the entire original data distribution. New studies (Han *et al.*, 2021a; Farhadi and Yang, 2020) adopt it in online learning and show benefits. Mullapudi *et al.* (2019) adopts online knowledge distillation to improve inference time for image segmentation. A light model is used for inference, and at fixed intervals, it extracts features from the selected frames (key frames) with an expensive model to train the light one. Instead of using fixed intervals, I (Farhadi and Yang, 2020) proposed (chapter 3) an adaptive mechanism to select key frames. Chapter 4 studies (Farhadi *et al.*, 2020) shows that processing the key frame on embedded devices will hurt the inference time and energy consumption. I extended ARGOS to online knowledge distillation without relying on key frame processing. Instead, I apply the expensive CNN model on RoIs, not the whole frame. The RoIs are detected based on light model output with low confidence. Moreover, a loss function is designed to use partial information extracted from RoIs combined with information extracted in early detection to train the light model.

5.2.4 Video Object Detection

Temporal cues have been used in video processing to reduce computational costs. Researchers adopt optical flow (Zhu *et al.*, 2017), or recurrent network (Liu and Zhu, 2018) architectures to modify the previous frame's extracted features and reuse

the features for current frame detection. These approaches heavily rely on the key frame mechanism, which is computationally expensive. Moreover, it can increase the response time when we have a new object in the scene. Liu *et al.* (2020) suggest the usage of external sensor data (LiDAR) to select regions for further processing; however, this information is not available in all applications. Here, ARGOS has an agile detector (pure vision-based) that detects RoIs and applies an expensive CNN model on them. Furthermore, it can retrain itself (based on implementation) to avoid future repetitive requests.

5.3 ARGOS Approach

The core of ARGOS is the decomposing image module, which locates RoIs in input frames and creates independent sub-regions for deep feature extraction. ARGOS can be implemented in active or passive mode. In passive mode, ARGOS detects RoIs as a pre-processing stage and reduces the DNN module's input size (and computation). I implemented this configuration using a binary region proposal. In active mode, Outputs from these regions can be aggregated with early detection to train the light detector (which also functions as the RoIs detector) and reduce the number of future operations in the online knowledge distillation methods. In the following, I describe these two main approaches to implement the ARGOS; 1) using a binary neural network (Passive), 2) using the light model in online knowledge distillation methods (Active).

5.3.1 Binary Region Proposal

Due to the memory and computation efficiency, Binary neural networks have received significant attention in recent years. However, these models cannot achieve high accuracy on complex problems such as object recognition. Therefore, I adapt

these models as a pre-processing step for proposing RoIs, which is a binary segmentation. Here, We need to find regions with a high probability of object existence. This task is much easier compared to semantic segmentation or object detection. I also tested different methods to reduce the computation of this model (the so-called "QU-Net").

Architecture: I adapted the U-net (Ronneberger *et al.*, 2015) architecture as the baseline. I divided the model into three parts with different quantization structures to develop a light version of U-Net. The backbone of the network used binary modules with the final layers using binary weights and 4-bit activations. Moreover, instead of using full precision on early layers, I quantized them to improve inference time without affecting accuracy.

For a convolutional neural network, the first layers are critical as losing any information cannot be reclaimed at later stages. Thus, it is essential to ensure that the relevant information is passed down with minimal disruption. I achieved this by using 1-bit weights and 8-bit activations in the convolution layers at the initial stage and the following two downsampling stages (the yellow regions in Fig. 5.1).

Once the initial layers extract the relevant features, the information capacity can be reduced with minimal loss of information. Therefore, the last two downsampling layers were replaced with 1-bit weights and 1-bit activations in QU-Net model (the gray regions in Fig. 5.1). My initial approach was to use Melius Blocks (Bethge *et al.*, 2020) which consist of a DenseBlock and Improvement Block that helps maintain the feature quality, especially with the increase of channels in the last few downsampling layers. However, it was not found to contribute significantly to the accuracy based on the experiments. Thus, I replaced the convolution layers with the DoReFa-Net based (Zhou *et al.*, 2016) binarized modules.

It was important for the relevant information to flow up to the final layer to

reconstruct the binary mask from the extracted features. The experiments showed that reducing the activation layer bandwidth to less than 4-bits led to severe accuracy degradation. Therefore, I use a 1-bit weight and a 4-bit activation to ensure that the information captured in the downsampling layers is fed through the upsampling layers (represented by dark green regions in Fig. 5.1) to the final feature map. The last layer is full-precision to provide a dynamic range of values that are obtained in the one-hot output tensor.

Each convolution layer with quantized weights and activations also has a squeeze and excitation block (Hu *et al.*, 2018). This approach can help in modeling channel-wise attention and increasing the information capacity of the model with a negligible increase in computation cost.

Forward and Backward Propagation: In QU-Net, the entire model is composed of 1-bit weights except for the last layer. The weights are binarized using the sign function that rounds each value to the closest integer in the set {-1,1} as defined in Eq. 5.1. To solve the issue of the non-differentiability of the sign function, the Straight-Through Estimator (STE) (Bengio *et al.*, 2013) was utilized which passes the output of the gradient as it is.

$$D_t(x, y) = \begin{cases} +1 & \text{if } x \geq 0 \\ -1 & \text{otherwise.} \end{cases} \tag{5.1}$$

Each weight parameter also has a multiplicative scalar constant associated with it that allows the increase in the range of weights while still utilizing bit convolution kernels. This technique led us to use a scalar constant for each weight which is equal to the mean of all the absolute values in the specific parameter as defined in Eq. 5.2. Zhou *et al.* (2016) first proposed this method which allowed the network the flexibility to use bit-convolutional kernels both in forward-propagation as well as back-propagation.

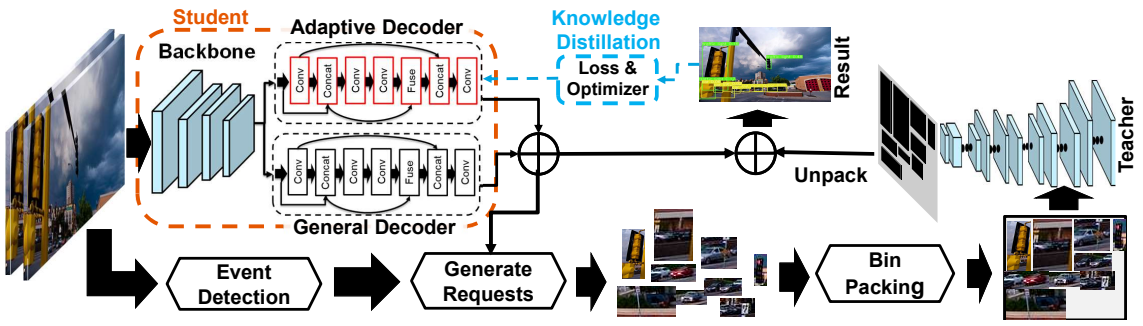

Figure 5.3: ARGOS Implementation with the Online Knowledge Distillation Method: A Light Student Model Is Responsible for Detecting Objects in the Scene in Addition to an Event Detection System to Extract the Regions of Interest. These Regions Are Packed to Obtain a Compact Input Containing Only the Regions of Interest, Which Is Then Sent to a Deeper Model for Final Detection. Further, the Light Model Is Trained Adaptively to Reduce the Dependency on the Feature Extraction from the Deeper Model.

The activation quantization methodology followed three different representations depending on the depth of the layer. The quantization functions as defined in Eq 5.2 and 5.3 adopted. Here r_i represents the number in the real format while r_o is the quantized version of the number with k representing the number of bit representation.

- **Forward Pass**

$$r_o = \begin{cases} sign(r_i) \cdot mean(abs(r_i)) & \text{if } weight \\ \frac{1}{2^k-1} round((2^k-1)r_i) & \text{if } activation \end{cases} \tag{5.2}$$

- **Backward Pass**

$$\frac{dl}{dr_i} = \frac{dl}{dr_o}. \tag{5.3}$$

Training: The training was performed for a standard 50 epochs. I used an RMSProp optimizer (Hinton *et al.*, 2012) with a learning rate of "1e-5", weight decay of "1e-8", and momentum of "0.9". I used a lower learning rate than usual to account for the small range of the weight values. The weights were binarized at forwarding pass, but each back-propagation step involved calculating the gradients on the real-valued weight tensors. All of the training was performed on images of resolution

320×320. Although the initial training was performed on 640×640 images, it was found that the reduction in accuracy was negligible, allowing us to use a lighter model. Using a lower resolution of 160×160 increased the area selected drastically, making it unfeasible for use as a region proposal network. For deeper networks, images of 640×640 resolution are used.

My focus in building the region proposal system was to ensure that all the objects were detected with very few missed objects. Therefore, I modified the loss function to provide more weights to the foreground class with a lesser focus on the background class. The loss function I used was a combination of weighted CrossEntropy + scaled Dice loss. This came at the expense of a larger number of false positives but allowed the capture of most of the regions, which are essential to a region proposal network.

$$\begin{aligned} L_{roi} = & -weight_{ce}[0] * log(exp(x[0])/(\sum_j exp(x[j]))) \\ & - weight_{ce}[1] * log(exp(x[1])/(\sum_j exp(x[j]))) \\ & sf_{dc}[0] * \frac{\sum p_0 g_0}{\sum p_0 + \sum g_0} + sf_{dc}[1] * \frac{\sum p_1 g_1}{\sum p_1 + \sum g_1}. \end{aligned} \quad (5.4)$$

Here, the cross-entropy losses are defined by the initial two parts of Eq. 5.4 where x represents the class probability. Each class (0,1) also has a corresponding weight $weight_{ce}$ associated with it, equal to the inverse number of samples in the class. The following two parts of the equation represent the dice coefficients for each class with a scale factor (sf_{dc}) associated with each class. The scale factor is the inverse ratio of samples summing to 1. Here p represents the predicted labels, and g represents the actual labels. It helps measure the overlap between the two sets and improves accuracy when combined with the cross-entropy loss.

Validation: The predicted mask consists of a 2-dimensional vector containing

the values for each class (binary). This is reduced to a one-dimensional vector where elements indicate the argument for the maximum score, with 0 being the background and 1 being the foreground. The final output is dilated to ensure the surrounding regions around the objects are covered, which is essential in visual recognition tasks such as segmentation where the surrounding context can help improve the accuracy. I focused on testing the validity of the algorithm on the two main metrics -

- The number of regions detected with an IOU threshold of 50%, 75%, and 95%;
- The amount of area covered by the true positives compared to the actual area of labels.

5.3.2 Online Knowledge Distillation

To show the generality of ARGOS, I extend it to online knowledge distillation models. In these models, there is a light (or shallow) model which adapts to the observed environment (Farhadi and Yang, 2020; Zhang and Ma, 2020). I swap the QU-Net with the light model in these types of feature extractors. Next, I take the low confidence output of the light model with an event detection mechanism to propose RoIs that need deep processing. The use of multiple feature extractors can improve the system response time while maintaining accuracy. I (Farhadi *et al.*, 2020) showed the benefits of this method, specifically on fixed cameras (similar to vIM), as the distribution of objects does not change significantly. I added the ARGOS mechanism to this methodology using the following steps. First, the light detector (student) plays the role of the initial object detector and region proposal (RoIs). Student output consists of two types of detections: 1) Objects with low confidence; 2) Objects with high confidence—objects with high confidence considered as accurate detections. Regions with low confidence detected objects selected as RoIs. I combined this with

a background-foreground subtraction (**camera is fixed**) to select RoIs that I have also observed an event to improve the performance. Next, I send the RoIs for deep feature extraction using the deeper model (teacher). Using the extracted feature by the teacher, I update the student to avoid future requests. In Fig.5.3, I depict these parts.

Event Detection

To identify the challenging regions (RoIs), I adopt a decision mechanism with motion detection; specifically, a Forgetting Morphological Temporal Gradient (FMTG) (Richefeu and Manzanera, 2006). FMTG implements a nonlinear $\Sigma\Delta$ filter, which is known for an efficient analog to digital conversion. Formally, I compute the current background image M_t and the time-variance image V_t iteratively:

- Compute mean over incoming frames, $I_0, ..., I_{t-1}, I_t$:

$$
\begin{aligned}
&M_0(x,y) = I_0(x,y),\\
&M_t(x,y) = M_{t-1}(x,y) + sgn[I_t(x,y) - M_{t-1}(x,y)],\\
&\Delta_t(x,y) = |M_t(x,y)|.
\end{aligned} \tag{5.5}
$$

- Compute the variance:

$$
\begin{aligned}
&V_0(x,y) = \Delta_0(x,y),\\
&if\ \ \Delta \neq 0, V_t(x,y) = V_{t-1}(x,y) +\\
&\qquad sgn[N \times \Delta_t(x,y) - V_(t-1)(x,y)].
\end{aligned} \tag{5.6}
$$

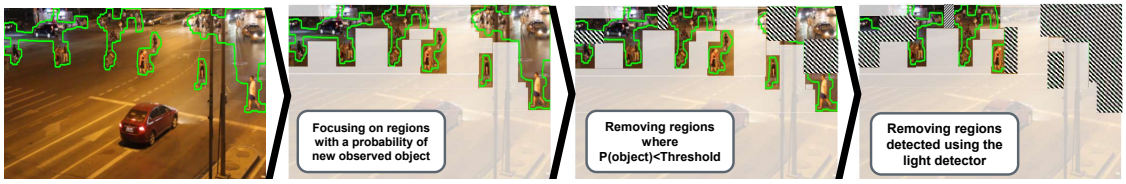

Figure 5.4: Original Image Vs Regions of Interest (RoIs); I Start with the Regions Proposed by the Event Detection Mechanism. These Are Further Reduced to Remove the Regions with No Chance of Object Existence and Regions with Pre-detected Objects.

- Motion label:

$$D_t(x,y) = \begin{cases} 0 & \text{if } \Delta_t(x,y) < V_t(x,y) \\ 1 & \text{otherwise.} \end{cases} \tag{5.7}$$

Here, N denotes an amplification factor for new incoming frames. This procedure can produce small noisy regions inside selected regions (false negatives) and also not selected regions (false positives), which I call bubbles. Therefore, I apply a dilation filter on the output, followed by an erosion filter.

Further Processing Requests Generation

After the initial detection and RoIs, I locate areas that need further processing. As shown in Fig. 5.4, I segment the image into sub-regions and focus on those regions which have activity based on event detection. Next, I remove regions with a low chance of object existence based on the general decoder's output, $Pr(object) < \Gamma$, where the Γ could be tuned based on the desired recall on the validation set during training time. The experiments showed that the adaptive decoder loses its generality at inference time; hence I use the general decoder for region selection. Finally, I check the "intersection over union" (IOU) of these regions with objects the adaptive decoder has detected earlier. If the adaptive decoder already detected objects, I remove them from future processing.

Knowledge Distillation Stage

In this stage, ARGOS uses the consolidated results from the previous module to improve the light detector or student. This action will reduce future requests (in-depth processing) and improve the ARGOS response time by detecting most of the objects using the student model. Back-propagation based training relies on a loss function comparing the student and teacher output. As the consolidated results have been produced by combining the student and the teacher, I do not have access to teacher output over the whole frame; consequently, a new loss function is needed to train the student using partial knowledge.

Previously, I (Farhadi and Yang, 2020) adopt a combination of student's and teacher's outputs as the ground truth for supervised student re-training, to avoid a sharp shift of the student model towards teacher's output and thus preserve student's knowledge by the following loss function:

$$\begin{aligned} L_{final} &= \sum \|T_s^H - T_t^H\|_2^2 \\ &+ \sum \|T_s^L - ((\lambda * T_s^L) + ((1-\lambda) * T_t^E))\|_2^2, \end{aligned} \tag{5.8}$$

where λ denotes the modulation factor in their weighted loss function. T_s^H and T_t^H are the students and teacher output tensors with high confidence in object existence and T_s^L and T_t^L tensors with low confidence. This loss function needs the teacher and student to have a similar structure (single-shot).

To have a detection-model-agnostic system, the loss function should be general enough to work with any object detection model as a teacher (single-shot or two-stage detector). Second, the loss function should maintain the current knowledge of students while distilling new information. To this end, I present a loss function that fulfills the design requirements. First, I calculate the following losses:

Localization Loss:

$$
\begin{aligned}
&L_{loc} = \frac{1}{|\mathbb{D}|}\sum_{\mathbb{D}}(1 - GIOU(B^S_{i,j}, B^T_{i,j})) + \frac{1}{|\mathbb{N}|}\sum_{\mathbb{N}}\|B^S_{i,j} - B^{SI}_{i,j}\|^2_2, \\
&where,\ B_{i,j} = (x, y, w, h), C_{i,j} = (c_1, ..., c_n), 0 \leq O_{i,j} \leq 1, \\
&\mathbb{D} = \{\alpha | \alpha\ is\ a\ region\ processed\ by\ teacher.\}, \mathbb{N} = \{\alpha | \alpha \notin \mathbb{D}\}.
\end{aligned}
\tag{5.9}
$$

Classification Loss (with Binary Cross-Entropy (BCE):

$$
\begin{aligned}
&L_{cls} = \frac{1}{|\mathbb{D}|}\sum_{\mathbb{D}} BCE(C^S_{i,j}, C^T_{i,j}) + \frac{1}{|\mathbb{N}|}\sum_{\mathbb{N}}\|C^S_{i,j} - C^{SI}_{i,j}\|^2_2, \\
&where,\ \sigma(x) = e^x/(e^x + 1), \\
&BCE((x_1, ..., x_n), (y_1, ..., y_n)) = \\
&\sum_n -w_i[y_i.log(\sigma(x_i)) + (1 - y_i).log(1 - \sigma(x_i)))].
\end{aligned}
\tag{5.10}
$$

Here, w is for weighting classes in an unbalanced dataset.

Objectness Loss (with a generalized intersection over union loss (GIOU) (Rezatofighi *et al.*, 2019)):

$$
L_{obj} = \frac{1}{|\mathbb{D}|}\sum_{\mathbb{D}} BCE(O^S_{i,j}, O^T_{i,j}) + \frac{1}{|\mathbb{N}|}\sum_{\mathbb{N}}\|O^S_{i,j} - O^{SI}_{i,j}\|^2_2. \tag{5.11}
$$

B, C, and O are the box coordinates, class label, and object existence probability for each cell in the student's output tensor. I present student output before training as B^{SI}, C^{SI}, and O^{SI}. The student's output after each iteration is denoted as B^S, C^S, O^S, and the teacher's as B^T, C^T, O^T.

I calculate the overall loss by summing the localization, objectness, and classification loss in each training iteration: $Loss = \lambda_{loc}L_{loc} + \lambda_{obj}L_{obj} + \lambda_{cls}L_{cls}$ where λ_{loc}, λ_{obj}, and λ_{cls} are modulation factors. This overall loss function can take the ground truth from any object detector as a teacher as it takes detection coordinates and class for training.

5.3.3 Masking

RoIs can be processed separately or bached. I aim to process these requests using a GPU accelerator as they are efficient in performing batch processing. The proposed regions (the output mask of RoIs detector) could be processed in two fashions: 1) bin-packing; 2) Gather and scatter.

First, I adopt a bin-packing algorithm to put the regions (RoIs) next to each other, which is different from a conventional batching method due to the various sizes of the "proposed regions". The bin packing problem for two-dimensional objects is NP-hard. Therefore, I adopt the "Maximal rectangles best short side fit", a heuristic algorithm (Jylänki, 2010). Using this heuristic approach reduces the execution time for packing, which is negligible compared to the input frame's processing time through the teacher model. Furthermore, in the experiments, ARGOS can accommodate dynamic bin sizes. Therefore, if the teacher model could also accept dynamic input sizes, it could significantly reduce the computation.

The *gather-scatter* method can also be used to perform efficient tensor computation. The *gather* step takes each active position in the tensor and consolidates it to a smaller tensor, after which the normal convolution is applied. This is mapped back to the original tensor using the *scatter* method. Thus, it is effectively equal to performing operations on a smaller image based on the number of active positions with adding less overhead to the overall operation. Although this method focuses on the active regions with no outside areas processed, additional effort is required to implement a custom layer.

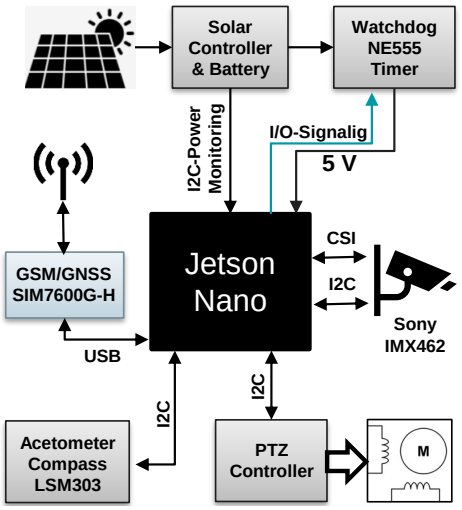

Figure 5.5: ARGOS System Design.

5.4 Experiments

To validate the framework, I conducted different experiments to compare ARGOS with state-of-the-art methods. In the following, I presented the experiments in two separate configurations; 1) Based on the Binary neural networks, 2) Based on online knowledge distillation.

5.4.1 Experiment Setup

Hardware: The embedded device is the NVIDIA Jetson Nano developer kit (jet, 2019), a quad-core ARM A57 CPU (operating at 1.43 GHz), 4 GB 64-bit LPDDR4 RAM, and 128-core Maxwell GPU. I also assigned a 2 GB swap memory to accommodate the memory overflow that can occur while testing heavy DNNs such as SpotNet (Perreault *et al.*, 2020) as the memory is shared between CPU and GPU. I measured the power consumption using tegrastats (jet, 2019). The training and other experiments were conducted on a Dell workstation with an Intel Xeon W-2125 CPU and an NVIDIA Titan Xp.

I have implemented the prototype version for vIM tests (Fig. 5.2). Fig. 5.5 shows

Table 5.1: Results of Quantizing Different Layers of QU-Net and the Effect of Applying a DCT-II Transform to the Input. Each Combination Was Trained for 50 Iterations. The mAP Is Calculated on the Instance Segmentation Classes of Cityscapes Dataset. The DCT Based Model Has a Lower Computation, but It Selects More than 99% of the Image, Which Does Not Reduce the Computation of the Deeper Model.

DCT Input	**Intial Layers Quantized (8bit/1bit)**	**Middle Layers Quantized (1bit/1bit)**	**Upsampling Layers Quantized (4bit/1bit)**	**RoIs Detected(%)**	**Area Processed**	**FLOPs(G)**
No	No	No	No	98.36	0.3658	62.68
No	No	Yes	No	98.23	0.3689	55.27
No	No	Yes	Yes	98.23	0.3870	18.36
No	**Yes**	**Yes**	**Yes**	**97.97**	**0.3815**	**5.30**
Yes	Yes	Yes	Yes	99.66	0.9820	0.112

the system design for the vIM implementation. A solar panel powers the visual system. The IMX462 camera observes the scene, and detected objects are sent to the server using NB-IoT network (Sinha *et al.*, 2017). In addition, I have included a watchdog for checking system low-level functionality, an accelerometer/compass, and a GNSS sensor used for localizing the field of view.

Dataset: I evaluated the Binary proposal model or QU-Net on the Cityscapes (Cordts *et al.*, 2016) which focuses on vehicle object detection and semantic understanding of urban street scenes. It consists of 3475 fine annotated images for train and validation sets. I split the cityscape dataset into 2975 training images and 500 validation images. Apart from that, I also trained the model on COCO dataset (Lin *et al.*, 2015) to test the performance on a large number of classes. The 2017 part of the COCO was used, which consists of 118k training and 5k validation images.

I also evaluated the online knowledge distillation (ARGOS version) on UA-DETRAC (Wen *et al.*, 2020) and WiseNet (Marroquin *et al.*, 2019) datasets. I selected these datasets as the camera is stationary, and I can use the event-based mechanism. UA-DETRAC has 140,000 frames of real-world traffic scenes. In this dataset, 1.2 million vehicles are labeled with bounding boxes. The videos were recorded at 25 frames per second (fps) in the JPEG format, and the resolution of images is 960×540 pixels. I

also conducted experiments on the WiseNet dataset to study the performance of my proposed design for indoor scenarios. The WiseNet dataset is composed of 62 videos recorded using six cameras for people detection and tracking.

Deep Learning platform: I perform all the experiments using PyTorch library (Paszke *et al.*, 2019). I use the same library for evaluating the performance of other methods.

5.4.2 Binary Region Proposal

I tested the BNN model (QU-Net) for RoIs proposal on different configurations to evaluate the effect of other compressing mechanisms on my model. I used the gather-scatter approach to apply the mask for the experiments on QU-Net. Although the binary region proposal model can result in increased computational complexity, the experiments showed that it is very minimal compared to the actual complexity of the deeper models. Thus, applying it to the image can reduce the overall computation significantly.

Quantization and DCT-based approach

I performed initial experiments to analyze two different methods to reduce the computation of a network -

- The effect of quantization on the U-Net network;
- The effect of using a 2D type II discrete cosine transform on the image to obtain a downsampled version that can be used as input to the network.

Quantization based approach: In CNNs, the main contributor to the complexity is the convolution operations. These operations have a significant overhead due to

Table 5.2: Results of the Object Detection Metrics on the Cityscapes and the COCO Dataset. Rn-101 Represents the Resnet-101 Backbone and Dc Represents the Dynamic Convolution Approach. I Use the QU-Net Model under the ARGOS Ecosystem as the Region Proposal Model in These Experiments.

Model	**Dataset**	**Precision**	**Recall**	**mAP@0.5**	**mAP@0.5:0.95**	**FLOPs(G)**
RN101-YOLO	Cityscapes (CityPersons)	68.7	49.5	56.4	32.5	42.29
DC based RN101-YOLO	Cityscapes (CityPersons)	66.2	45.5	51.1	27.2	27.58
RN101-YOLO + ARGOS	**Cityscapes (CityPersons)**	**69.5**	**45.9**	**52.9**	**30.1**	**21.93**
YOLOv5m	Cityscapes	76.6	46.5	53.3	31.1	51.41
YOLOv5m + ARGOS	Cityscapes	78.4	45.1	52	29.7	22.32
YOLOv5m	COCO	69.7	57.5	62.8	43.5	51.41
YOLOv5m + ARGOS	COCO	72.4	53.5	60.0	40.2	42.42
YOLOv5l	COCO	72.8	61.5	66.5	47.2	115.61
YOLOv5l + ARGOS	COCO	72.5	59	63.7	43.8	88.78
YOLOv5x	COCO	74.9	62.7	68.4	49.5	219.02
YOLOv5x + ARGOS	COCO	74.1	60.5	65.5	45.8	163.49

the repeated multiplication functions and heavy memory communications. Binarization can alleviate this overhead at the expense of accuracy drop. To understand how the binarization of each module affects accuracy, I performed various experiments as shown in Table 5.1. The quantized model reduced computation by a factor of 10 when compared to the full precision model. Moreover, the model occupied a space of 2MB, which allows us to fit the model on various low-memory devices. It can also be seen that the quantized model detects a close percentage of RoIs as the full-precision model, with a minor increase in the total area of the image captured.

DCT based approach: Along with quantization, to further reduce the computation cost, I include a DCT based method to reduce the computations of the overall network further. Xu et al. reported that applying DCT can reduce the computation cost of CNN models (Xu *et al.*, 2020). However, as reported in table 5.1, the experiments showed that a DCT-based approach for QU-Net did not bring any benefits in reducing the computation of deeper models. The output of the DCT based region proposal model did not conform to the actual object presence locations, which resulted in a large area being captured on dilating the outputs. The DCT model proposes 99% of the images as the regions of object presence (RoIs), equivalent to sending the entire image through the deeper network. Comparatively, the non-DCT approach selected only 38% of the images.

I tested ARGOS on two main applications of computer vision - object detection and segmentation to observe its effect on the accuracy and computation of proposed methods in these domains. I also compared the method with dynamic convolution (Verelst and Tuytelaars, 2020). Dynamic convolution tries to reduce the processing regions on residual neural networks such as ResNet. This type of convolution neural network has a computational budget parameter that can be set which determines the relative amount of FLOPs that should be executed. For example, a value of 0.25

indicates 25% of the FLOPs should be executed. To have a fair comparison, I used a value of 0.25 as the budget in all my experiments. The method is shown to outperform the dynamic convolution in both cases.

Object Detection: As mentioned, the dynamic convolution requires a model implemented using a residual backbone. Hence, I used a YOLO model with a ResNet-101 (Yang, 2021) backbone as the baseline model (the model can be plugged into any prebuilt framework). QU-Net was trained on the Cityscapes dataset separately for 50 epochs. Both the baseline and the dynamic convolutions (with a 0.25 budget) were trained for 200 epochs. Models were evaluated on the person detection annotations (CityPersons) as provided on the official website (cit, 2021). Table 5.2 shows the outperformance of ARGOS compares to the dynamic convolution approach. A combination of my model with the baseline model enabled a reduction of FLOPs by 50%. It was 6 GFLOPs lower than the dynamic convolutions along with an mAP gain of close to 3%. Moreover, the reported FLOPs are the computation cost of both the deeper model and QU-Net.

Segmentation: I used the PSPNet model with the ResNet-101 backbone as the baseline model. ARGOS model was trained on the Cityscapes dataset separately for 50 epochs. The baseline model and the dynamic convolution with a 0.25 budget were trained for 120 epochs each. This was evaluated on the instance segmentation classes in the Cityscapes dataset. Fig. 5.6a shows the method outperforming the dynamic convolution with an mAP gain of close to 20%. ARGOS has a computational reduction of 62%, whereas the dynamic convolution achieved a reduction of 41%.

COCO Dataset

To further understand the scalability of ARGOS, I performed experiments on the COCO dataset using the YoloV5 (CNN) model and the Swin(Transformer) model.

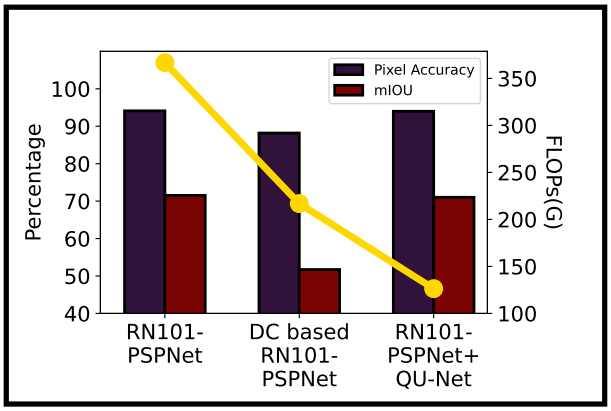

Figure 5.6: Results of the Segmentation Metrics on the Instance Segmentation Classes in the Cityscapes Dataset. Rn-101 Represents the Resnet-101 Backbone, and Dc Represents the Dynamic Convolution.

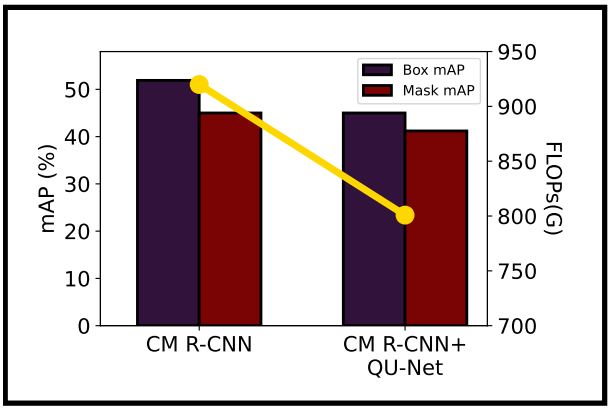

Figure 5.7: Results of the mAP Statistics for Object Detection and Segmentation Were Calculated on COCO Dataset. CM R-CNN Represents the Cascade Mask R-CNN.

These experiments show how ARGOS performs when I have more classes than the Cityscape dataset tests. I used the original models as the baseline and evaluated the performance with the QU-Net model, and without it.

The comparison between the Yolov5 and QU-Net + Yolov5 model is shown in Table 5.2. There was a significant reduction for the Cityscapes dataset (57%) with a slight reduction in model accuracy. For the COCO dataset, the decrease was less pronounced (25%). For the Swin transformer (Fig.5.6b), the method reduced the

number of FLOPs by 114 GFLOPs with a 4.8% reduction in object detection accuracy and 3.8% in instance segmentation accuracy.

The COCO dataset consists of images where most of the objects were captured at a close range. This increases the total area occupied by the objects when compared to other real-world datasets such as Cityscapes. My model exploits the real-world assumption of objects occupying a smaller area compared to the background. Thus, the reduction in computational cost is larger when I test the model on the Cityscapes dataset compared to the COCO dataset.

5.4.3 Online Knowledge Distillation

I (Farhadi *et al.*, 2020) showed that online knowledge distillation could reduce the computation of convolutional neural networks using temporal adaptation to the observing environment. However, their experiments showed that the key frame selection and processing are expensive for embedded devices and need to be processed on the cloud (Farhadi *et al.*, 2020). Therefore, I tested ARGOS approach for this type of CNNs to validate its effectiveness. In addition, I conducted the experiments on Jetson Nano to study ARGOS performance in a constrained environment.

UA-DETRAC Dataset

Training: I pre-train the student model, and teacher on COCO (Lin *et al.*, 2015) dataset. Then, the pre-trained models were fine-tuned on the UA-DETRAC dataset (offline training). The bag of freebies (Bochkovskiy *et al.*, 2020) is used to improve the accuracy during the offline training stage. In Knowledge distillation and training, the ADAM optimization (Kingma and Ba, 2014) was applied for end-to-end training. The optimization is performed in the 16-bit numerical format at the inference time. To improve the performance in 16-bit precision, I add 1e-4 to the denomina-

Table 5.3: Different Methods Have Been Compared in Terms of Accuracy (mAP), Frame per Second Obtained Using Workstation and Embedded Devices, Energy, and Score ($mAP/Energy$). ARGOS Improves the Energy Consumption by **89%** in Comparison with Yolov5x.

Model	Overall % ↑	Easy % ↑	Medium % ↑	Hard % ↑	Cloudy % ↑	Night % ↑	Rainy % ↑	Sunny % ↑	FPS ↑ WS	FPS ↑ Embedded	Energy (J) ↓	Score ↑
FG-BR_Net (Fu *et al.*, 2019)	79.96	93.49	83.60	70.78	87.36	78.42	70.50	89.8	10	-	-	-
HAT (Wu *et al.*, 2019a)	78.64	93.44	83.09	68.04	86.27	78.00	67.97	88.78	3.6	-	-	-
GP-FRCNN (Amin and Galasso, 2017)	77.96	92.74	85.39	67.22	83.23	77.75	70.17	86.56	4	-	-	-
SpotNet (Perreault *et al.*, 2020)	86.80	**97.58**	**92.57**	76.58	89.38	**89.53**	80.93	**91.42**	14	0.068	100.93	0.008
SSD-VDIG (Qijie and Feng, 2019)	82.68	94.60	89.71	70.65	89.81	83.02	73.35	88.11	2	0.092	41.28	0.020
YOLOv5x (Jocher *et al.*, 2021)	**86.89**	95.96	90.17	**79.89**	90.67	84.86	**83.42**	90.92	9	0.2	31.255	0.026
Yolov3+FP (Kim *et al.*, 2019)	85.29	96.04	89.42	76.55	88.00	88.67	78.90	88.91	9	0.395	15.975	0.050
TKD (Farhadi and Yang, 2020)	51.29	54.48	53.65	48.47	62.79	41.87	50.91	47.69	**47**	0.79	8.53	0.060
ARGOS	83.53	93.16	86.34	76.71	**90.71**	78.45	77.64	90.6	25	**1.8**	**3.44**	**0.22**

Figure 5.8: An Example of a Fast-moving Object Which Can Be Missed by Object Detectors Due to Their Inference Time.

tor (backpropagation stage) to improve numerical stability during online knowledge distillation.

Evaluation Metrics: The evaluation metric for accuracy in the UA-DETRAC detection benchmark is stringent: the Mean Average Precision (mAP) with a high Intersection over Union (IOU) threshold set to "0.7". As the target is embedded devices, both the inference time and energy need to be considered in evaluations. I study the inference time (FPS) and its effect on real-scenarios accuracy. The ratio of mAP to energy consumption (Alyamkin *et al.*, 2018) is used to evaluate the overall efficiency in terms of accuracy and energy consumption. I have ranked the proposed methods using this metric in Table5.3.

Implementation: ARGOS executes at 16-bit precision, and experiments are performed on the workstation and embedded environment. I adapted Yolov5x as the teacher in the experiments. Image resolution is set to 864×864 at inference time. I used Bin-packing for applying the mask. The online knowledge distillation is executed on a separate thread; event detection and early detection are running concurrently to improve inference time. To have a fair comparison in the target device (Jetson Nano), I transfer the state of art methods to 16-bit precision and PyTorch.

Discussion: Table 5.3 compares the proposed ARGOS with different models evaluated on UA-DETRAC. The ARGOS achieves the first rank among tested meth-

ods in terms of energy and accuracy score. Although I added extra steps such as event detection, the experiments showed that the region removal procedure reduces the total execution time. I reduced the processing time by 80 ms, as the event detection and light detector run in parallel in CPU and GPU (Event detection: 80ms, Backbone:237ms, Decoders:8.5ms). The experiments also showed that bin-packing and region-removal execution time are negligible compared to other modules (2.5ms 8.5ms). Thus, ARGOS could maintain accuracy while reducing the inference time in embedded devices.

I observe the low performance of SpotNet(Perreault *et al.*, 2020) in the embedded environment, which is due to the high memory requirements of this model (Although it was running on 16-bit precision). Due to limited memory space, FG-BR Net (Fu *et al.*, 2019), HAT (Wu *et al.*, 2019a), (Kouris *et al.*, 2019), GP-FRCNN (Amin and Galasso, 2017), were not tested on the target device. The feature extractors in these works are expensive, although they have reduced the second stage processing. TKD (Farhadi and Yang, 2020) (base on online knowledge distillation) has inferior performance due to the teacher's execution cost, which does not allow the system to be updated based on changes in the scene. This phenomenon shows the poor performance of methods that relies on the key frame mechanism in a constrained system.

In real-world scenarios for vIM, the system needs to know object types and their locations. Hence, we have detection and then tracking. If we detect an object, we can start following it with less computation. Considering this fact, I tried to understand what will happen if an object remains in the scene less than the object detector's execution time. Fig. 5.8 shows an example of this case where the camera (traffic camera) observed the marked car less than 3 seconds. In this example, most of the proposed methods will miss the object. I conducted another experiment to reevaluate

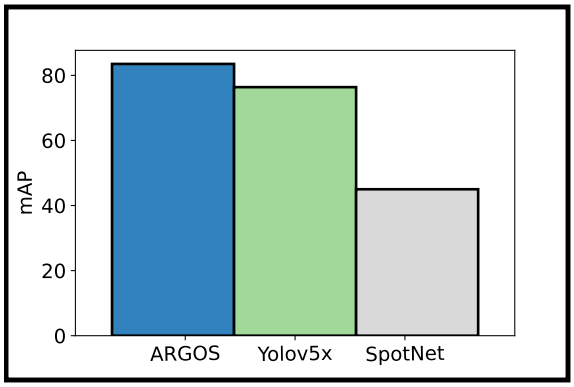

Figure 5.9: Results of the mAP Statistics Calculated on the UA-DETRAC Dataset with the Models Constrained to Process 25 Fps.

proposed methods based on this fact. Here, I removed objects which are in the scene less than the inference time of each method and ran the evaluation on their results. I removed objects using tracking information in the UA-DETRAC dataset. Fig. 5.9 shows the final results, which demonstrates the effectiveness of ARGOS in real-world scenarios.

WiseNet Dataset

To further validate ARGOS performance in fixed camera scenarios (for vIM), I designed another experiment in indoor videos using the WiseNet dataset (Marroquin *et al.*, 2019). Here, I understand the benefit of early detection and online knowledge distillation on system inference time and accuracy. I compare ARGOS with its Teacher (Yolov5x) and a modified ARGOS version, which doesn't have the online knowledge distillation and light detector module. All these configurations have been tested on the target embedded device in 16-bit precision at 512×512 resolution. Fig. 5.10 shows the Mean Average Precision (mAP) with $IOU = 0.5$ and inference time of mentioned configurations. Although ARGOS has not reached its teacher accuracy; however, its inference time is 4.9× less than the teacher. Moreover, ARGOS modified

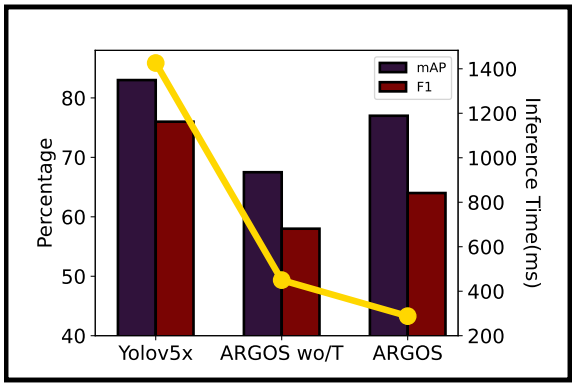

Figure 5.10: Results on the Wisenet Dataset. The Results Listed Show the Execution Time and Accuracy of the Model with Online Knowledge Distillation and Without It (wo/T).

version's (wo/T) accuracy is less than ARGOS and its inference time is significantly more than ARGOS, which shows the benefits of online knowledge distillation. The knowledge distilled from the deeper model to the light model (student) helps in reducing the number of requests to be processed by the teacher model.

5.5 Conclusion

This chapter presents an approach for efficient temporal knowledge distillation. This approach focuses on regions with a probability of object existence and more complexity and extracts deep features from them. I showed that this approach could reduce inference time and make the running oracle model easier in an embedded system. I tested this concept over different types of deep neural networks (DNNs) and showed its superiority over running the DNN on the whole frame.

Chapter 6

CONCLUSION

Throughout the dissertation, I have studied different aspects of an adaptive system for processing images on an embedded system with limited resources. In addition, I studied the applications' characteristics and their effect on system performance. Finally, I concluded this dissertation by proposing ARGOS for processing images efficiently and testing it on traffic monitoring applications.

In chapter 2, I proposed a new approach to running heavy neural networks on a platform based on FPGAs (with reconfiguration capability) with constrained resources. I stacked various shallow and deep models for quantized neural networks yielding an adaptive and hierarchical structure. This adaptive structure uses partial reconfiguration in FPGA to adapt itself based on the complexity of the input image and extract deep features if needed. I also designed a feedback procedure that controls the hardware adaption. To validate its performance, I tested the proposed system (AH-CNN) on CIFAR-10, CIFAR-100, and SVHN. The experiments showed that this structure could improve inference time while maintaining accuracy. AH-CNN is suitable for applications that need adaptive behavior towards dynamic priority change over object categories.

Although AH-CNN can reduce inference time, the number of hardware adaptions can still affect inference time if it happens frequently. I solved this issue by designing a novel approach to distilling temporal knowledge of an accurate but slow object detection model to a tinier model. In chapter 3, I suggested using previous observations and shifting the shallow model knowledge to the observing domain. Hence, I was able to reduce the deep feature extraction on the next images. I called this method tem-

poral knowledge distillation or TKD. I tested TKD on the Hollywood scene dataset, Youtube object dataset, the pursuit of happiness movie, and the office TV series. The experiments showed that TKD maintains a high inference efficiency while achieving a high recognition accuracy. Furthermore, I observed that the accuracy of TKD can reach the Oracle model in specific applications.

In chapter 5, I studied the temporal knowledge distillation concept in a constrained system. I tested the system in different system designs. Finally, based on the observations, I introduced a framework that can facilitate temporal adaptation in an embedded environment with limited resources.

I introduced and implemented a framework for incremental knowledge transfer, which can rely on local or cloud knowledge based on network latency. The parameters of a shallow model running on the user-end device are updated on a cloud clone or local instance during inference at some key frames to achieve close accuracy to the oracle model. I tested the proposed approach in different real-world scenarios where the camera can be fixed or moving. This framework resulted in 78% energy reduction when compared to running the deep model alone. However, the experiments also revealed that communication latency or the number of key frames could substantially affect vision adaptation.

I find out that the key frame selection procedure and training over the whole key frame are not efficient, and there are several concerns:

- There is a potential that a new object is added to the environment, and the key frame selection does not detect this change on time.

- The key frame selector chooses unnecessary frames for retraining and increasing the inference time.

- The temporal adaptation module passes the whole key frame to the oracle while

the system has uncertainty over a small region, not the entire frame.

In chapter 5, I introduced a novel approach that, by focusing on the region of interests, reduces the knowledge transfer significantly and addresses mentioned issues. I present an approach for decomposing images into independent sub-regions and reducing the processing units. Although BNNs cannot reach state-of-the-art accuracy, I showed that these networks could significantly reduce the computation of DNNs while maintaining accuracy. Moreover, ARGOS can be extended to other methods to improve the processing speed while maintaining reasonably high accuracy and, at the same time, saving energy. Experiments on traffic camera videos (vIM application) showed 89% energy reduction and 4.9× faster inference speed. It shows the sparsity of objects in vIM compared to other applications in which objects are close to the camera (coco dataset). Furthermore, the experiments reveal the effectiveness of online training and its positive effect on efficiency. Finally, the promising experimental results suggest two potential future research avenues: 1) schedule the sub-process (RoIs) in a heterogeneous environment; 2) distill knowledge to the student model with partial information while processing other regions.

REFERENCES

"2018 system design contest", http://www.cse.cuhk.edu.hk/ byu/2018-DAC-SDC/index.html (2018).

"Jetson nano developer kit (online)", https://developer.nvidia.com/embedded/jetson-nano-developer-kit, accessed: 2019-10-12 (2019).

"The cityscapes dataset", https://www.cityscapes-dataset.com/, accessed: 2021-08-13 (2021).

Al Kadi, M., P. Rudolph, D. Gohringer and M. Hubner, "Dynamic and partial reconfiguration of zynq 7000 under linux", in "2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig)", pp. 1–5 (IEEE, 2013).

Alyamkin, S., M. Ardi, A. Brighton, A. C. Berg, Y. Chen, H.-P. Cheng, B. Chen, Z. Fan, C. Feng, B. Fu *et al.*, "2018 low-power image recognition challenge", arXiv preprint arXiv:1810.01732 (2018).

Amin, S. and F. Galasso, "Geometric proposals for faster r-cnn", in "2017 14th IEEE AVSS", pp. 1–6 (IEEE, 2017).

Ba, J. and R. Caruana, "Do deep nets really need to be deep?", in "Advances in neural information processing systems", pp. 2654–2662 (2014).

Badrinarayanan, V., A. Kendall and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation", CoRR **abs/1511.00561**, URL http://arxiv.org/abs/1511.00561 (2015).

Bengio, E., P.-L. Bacon, J. Pineau and D. Precup, "Conditional computation in neural networks for faster models", arXiv preprint arXiv:1511.06297 (2015).

Bengio, Y., N. Léonard and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation", (2013).

Bethge, J., C. Bartz, H. Yang, Y. Chen and C. Meinel, "Meliusnet: Can binary neural networks achieve mobilenet-level accuracy?", arXiv preprint arXiv:2001.05936 (2020).

Bochkovskiy, A., C.-Y. Wang and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection", arXiv preprint arXiv:2004.10934 (2020).

Bolukbasi, T., J. Wang, O. Dekel and V. Saligrama, "Adaptive neural networks for efficient inference", in "International Conference on Machine Learning", pp. 527–536 (2017).

Bomey, N., "Deaths from cars running red lights hit 10-year high, aaa study finds", URL https://www.usatoday.com/story/money/2019/08/29/traffic-deaths-red-light-running-aaa-study/2122242001/ (2019).

Breuers, S., L. Beyer, U. Rafi and B. Leibel, "Detection-tracking for efficient person analysis: The detta pipeline", in "2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)", pp. 48–53 (IEEE, 2018).

Cai, H., L. Zhu and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware", in "International Conference on Learning Representations", (2019), URL `https://arxiv.org/pdf/1812.00332.pdf`.

Carrio, A., S. Vemprala, A. Ripoll, S. Saripalli and P. Campoy, "Drone detection using depth maps", in "2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)", pp. 1034–1037 (IEEE, 2018).

Castellano, B., "Pyscenedetect", (2018).

Chan, F.-H., Y.-T. Chen, Y. Xiang and M. Sun, "Anticipating accidents in dashcam videos", in "Asian Conference on Computer Vision", pp. 136–153 (Springer, 2016).

Chen, C., A. Seff, A. Kornhauser and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving", in "Proceedings of the IEEE International Conference on Computer Vision", pp. 2722–2730 (2015a).

Chen, G., W. Choi, X. Yu, T. Han and M. Chandraker, "Learning efficient object detection models with knowledge distillation", in "Advances in Neural Information Processing Systems", pp. 742–751 (2017a).

Chen, L.-C., G. Papandreou, I. Kokkinos, K. Murphy and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs", IEEE transactions on pattern analysis and machine intelligence **40**, 4, 834–848 (2017b).

Chen, T. Y.-H., L. Ravindranath, S. Deng, P. Bahl and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices", in "Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems", pp. 155–168 (ACM, 2015b).

Chen, Y., W. Li, C. Sakaridis, D. Dai and L. Van Gool, "Domain adaptive faster r-cnn for object detection in the wild", (2018).

Chin, T.-W., R. Ding and D. Marculescu, "Adascale: Towards real-time video object detection using adaptive scaling", arXiv preprint arXiv:1902.02910 (2019).

Cichy, R. M., D. Pantazis and A. Oliva, "Resolving human object recognition in space and time", Nature Neuroscience **17**, 3, 455–462, URL `http://www.nature.com/doifinder/10.1038/nn.3635` (2014).

Cichy, R. M., D. Pantazis and A. Oliva, "Similarity-based fusion of meg and fmri reveals spatio-temporal dynamics in human cortex during visual object recognition", Cerebral Cortex **26**, 8, 3563–3579 (2016).

Clifford, C. W., M. A. Webster, G. B. Stanley, A. A. Stocker, A. Kohn, T. O. Sharpee and O. Schwartz, "Visual adaptation: Neural, psychological and computational aspects", Vision research **47**, 25, 3125–3131 (2007).

Cordts, M., M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth and B. Schiele, "The cityscapes dataset for semantic urban scene understanding", in "Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)", (2016).

Courbariaux, M., I. Hubara, D. Soudry, R. El-Yaniv and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1", arXiv preprint arXiv:1602.02830 (2016).

Dai, D. and L. Van Gool, "Dark model adaptation: Semantic image segmentation from daytime to nighttime", arXiv preprint arXiv:1810.02575 (2018).

Dai, J., Y. Li, K. He and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks", in "Advances in neural information processing systems", pp. 379–387 (2016).

Daniels, G., "The Office", (2013).

Dean, J., G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, "Large scale distributed deep networks", in "Advances in neural information processing systems", pp. 1223–1231 (2012).

Eshratifar, A. E., M. S. Abrishami and M. Pedram, "Jointdnn: an efficient training and inference engine for intelligent mobile cloud computing services", arXiv preprint arXiv:1801.08618 (2018).

Eshratifar, A. E. and M. Pedram, "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", pp. 111–116 (ACM, 2018).

Everingham, M., L. Van Gool, C. K. Williams, J. Winn and A. Zisserman, "The pascal visual object classes (voc) challenge", International journal of computer vision **88**, 2, 303–338 (2010).

Farhadi, M., M. Ghasemi, S. Vrudhula and Y. Yang, "Enabling incremental knowledge transfer for object detection at the edge", in "Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops", pp. 396–397 (2020).

Farhadi, M., M. Ghasemi and Y. Yang, "A novel design of adaptive and hierarchical convolutional neural networks using partial reconfiguration on fpga", in "2019 IEEE High Performance Extreme Computing Conference (HPEC)", pp. 1–7 (IEEE, 2019).

Farhadi, M. and Y. Yang, "Tkd: Temporal knowledge distillation for active perception", arXiv preprint arXiv:1903.01522 (2019).

Farhadi, M. and Y. Yang, "Tkd: Temporal knowledge distillation for active perception", in "2020 IEEE WACV", pp. 942–951 (IEEE, 2020).

Figurnov, M., M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. P. Vetrov and R. Salakhutdinov, "Spatially adaptive computation time for residual networks.", in "CVPR", vol. 2, p. 7 (2017).

Fu, Z., Y. Chen, H. Yong, R. Jiang, L. Zhang and X.-S. Hua, "Foreground gating and background refining network for surveillance object detection", IEEE Transactions on Image Processing **28**, 12, 6077–6090 (2019).

Glorot, X. and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks", in "Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics", pp. 249–256 (2010).

Gong, Y., L. Liu, M. Yang and L. Bourdev, "Compressing deep convolutional networks using vector quantization", arXiv preprint arXiv:1412.6115 (2014).

Gupta, S., S. Ali, L. Goldsmith, B. Turney and J. Rittscher, "Mi-unet: Improved segmentation in ureteroscopy", in "2020 IEEE 17th International Symposium on Biomedical Imaging (ISBI)", pp. 212–216 (2020).

Gupta, S., J. Hoffman and J. Malik, "Cross modal distillation for supervision transfer", in "Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition", pp. 2827–2836 (2016).

Han, D., D. Im, G. Park, Y. Kim, S. Song, J. Lee and H.-J. Yoo, "Hnpu: An adaptive dnn training processor utilizing stochastic dynamic fixed-point and active bit-precision searching", IEEE Journal of Solid-State Circuits (2021a).

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

Han, S., H. Mao and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding", in "Proceedings of the International Conference on Learning Representations", (2016a).

Han, S., J. Pool, J. Tran and W. Dally, "Learning both weights and connections for efficient neural network", in "Advances in Neural Information Processing Systems", pp. 1135–1143 (2015b).

Han, S., H. Shen, M. Philipose, S. Agarwal, A. Wolman and A. Krishnamurthy, "Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints", in "Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services", pp. 123–136 (2016b).

Han, Y., G. Huang, S. Song, L. Yang, H. Wang and Y. Wang, "Dynamic neural networks: A survey", (2021b).

He, K., X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition", in "Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition", pp. 770–778 (2016a).

He, K., X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition", 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) pp. 770–778, URL http://ieeexplore.ieee.org/document/7780459/ (2016b).

Hinton, G., N. Srivastava and K. Swersky, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude", COURSERA: Neural networks for machine learning **4**, 2, 26–31 (2012).

Hinton, G., O. Vinyals and J. Dean, "Distilling the knowledge in a neural network", stat **1050**, 9 (2015).

Hu, J., L. Shen and G. Sun, "Squeeze-and-excitation networks", in "Proceedings of the IEEE conference on computer vision and pattern recognition", pp. 7132–7141 (2018).

Iandola, F. N., S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size", arXiv preprint arXiv:1602.07360 (2016).

Isensee, F. and K. H. Maier-Hein, "Or-unet: an optimized robust residual u-net for instrument segmentation in endoscopic images", (2020).

Izadyyazdanabadi, M., E. Belykh, M. Mooney, N. Martirosyan, J. Eschbacher, P. Nakaji, M. C. Preul and Y. Yang, "Convolutional neural networks: Ensemble modeling, fine-tuning and unsupervised semantic localization", arXiv preprint arXiv:1709.03028 (2017).

Jocher, G., A. Stoken, Ayush Chaurasia, J. Borovec, NanoCode012, TaoXie, Yonghye Kwon, Kalen Michael, L. Changyu, Jiacong Fang, Abhiram V, Laughing, Tkianai, YxNONG, P. Skalski, A. Hogan, Jebastin Nadar, Imyhxy, L. Mammana, AlexWang1900, C. Fati, D. Montes, J. Hajek, L. Diaconu, M. T. Minh, , Marc, Albinxavi, , Fatih, , Oleg and Wanghaoyang0106, "ultralytics/yolov5: v6.0 - yolov5n 'nano' models, roboflow integration, tensorflow export, opencv dnn support", URL https://zenodo.org/record/5563715 (2021).

Jokic, P., S. Emery and L. Benini, "Binaryeye: A 20 kfps streaming camera system on fpga with real-time on-device image recognition using binary neural networks", in "2018 IEEE 13th International Symposium on Industrial Embedded Systems (SIES)", pp. 1–7 (IEEE, 2018).

Jylänki, J., "A thousand ways to pack the bin-a practical approach to two-dimensional rectangle bin packing", (2010).

Kalman, R., "A new approach to linear filtering and prediction problems", Journal of Basic Engineering **82**, 1, 35–45 (1960).

Kang, Y., J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge", in "ACM SIGARCH Computer Architecture News", vol. 45, pp. 615–629 (ACM, 2017).

Kästner, F., B. Janßen, F. Kautz, M. Hübner and G. Corradi, "Hardware/software codesign for convolutional neural networks exploiting dynamic partial reconfiguration on pynq", in "2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)", pp. 154–161 (IEEE, 2018).

Khayatian, M. *et al.*, "R2 im-robust and resilient intersection management of connected autonomous vehicles", in "ITSC", (IEEE, 2020).

Kim, K.-J., P.-K. Kim, Y.-S. Chung and D.-H. Choi, "Multi-scale detector for accurate vehicle detection in traffic surveillance data", IEEE Access **7**, 78311–78319 (2019).

Kim, Y.-D., E. Park, S. Yoo, T. Choi, L. Yang and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications", arXiv preprint arXiv:1511.06530 (2015).

Kingma, D. and J. Ba, "Adam: A method for stochastic optimization", arXiv preprint arXiv:1412.6980 (2014).

Kouris, A., C. Kyrkou and C.-S. Bouganis, "Informed region selection for efficient uav-based object detectors: altitude-aware vehicle detection with cycar dataset", in "2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)", pp. 51–58 (IEEE, 2019).

Krizhevsky, A. and G. Hinton, "Learning multiple layers of features from tiny images", (2009).

Krizhevsky, A., I. Sutskever and G. Hinton, "Imagenet classification with deep convolutional neural networks", in "NIPS 2012", (2013).

Lea, C., R. Vidal and G. D. Hager, "Learning convolutional action primitives for fine-grained action recognition", in "Robotics and Automation (ICRA), 2016 IEEE International Conference on", pp. 1642–1649 (IEEE, 2016).

Li, H., Z. Lin, X. Shen, J. Brandt and G. Hua, "A convolutional neural network cascade for face detection", in "Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition", pp. 5325–5334 (2015).

Lin, T.-Y., P. Goyal, R. Girshick, K. He and P. Dollár, "Focal loss for dense object detection", in "Proceedings of the IEEE international conference on computer vision", pp. 2980–2988 (2017).

Lin, T.-Y., P. Goyal, R. Girshick, K. He and P. Dollár, "Focal loss for dense object detection", IEEE transactions on pattern analysis and machine intelligence (2018).

Lin, T.-Y., M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick and P. Dollár, "Microsoft coco: Common objects in context", (2015).

Lin, T.-Y., M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár and C. L. Zitnick, "Microsoft coco: Common objects in context", in "Computer Vision–ECCV 2014", pp. 740–755 (Springer, 2014).

Liu, M. and M. Zhu, "Mobile video object detection with temporally-aware feature maps", in "Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition", pp. 5686–5695 (2018).

Liu, M., M. Zhu, M. White, Y. Li and D. Kalenichenko, "Looking fast and slow: Memory-guided mobile video object detection", arXiv preprint arXiv:1903.10172 (2019).

Liu, S., S. Yao, X. Fu, R. Tabish, S. Yu, A. Bansal, H. Yun, L. Sha and T. Abdelzaher, "On removing algorithmic priority inversion from mission-critical machine inference pipelines", in "2020 IEEE Real-Time Systems Symposium (RTSS)", pp. 319–332 (IEEE, 2020).

Liu, W., D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg, "Ssd: Single shot multibox detector", in "European conference on computer vision", pp. 21–37 (Springer, 2016).

Liu, Z., Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows", in "Proceedings of the IEEE/CVF International Conference on Computer Vision", pp. 10012–10022 (2021).

Lu, H., L. Zhang, Z. Cao, W. Wei, K. Xian, C. Shen and A. van den Hengel, "When unsupervised domain adaptation meets tensor representations", (2017).

Mao, J., Q. Yang, A. Li, H. Li and Y. Chen, "Mobieye: An efficient cloud-based video detection system for real-time mobile applications", in "Proceedings of the 56th Annual Design Automation Conference 2019", pp. 1–6 (2019).

Marroquin, R., J. Dubois and C. Nicolle, "Wisenet: An indoor multi-camera multi-space dataset with contextual information and annotations for people detection and tracking", Data in brief **27**, 104654 (2019).

Marszałek, M., I. Laptev and C. Schmid, "Actions in context", in "IEEE Conference on Computer Vision & Pattern Recognition", (2009).

Mehta, R. and C. Ozturk, "Object detection at 200 frames per second", arXiv preprint arXiv:1805.06361 (2018).

Mokri, S., "Fish and cat", (2013).

Muccino, G., "The pursuit of happyness", (2008).

Mullapudi, R. T., S. Chen, K. Zhang, D. Ramanan and K. Fatahalian, "Online model distillation for efficient video inference", arXiv preprint arXiv:1812.02699 (2018).

Mullapudi, R. T., S. Chen, K. Zhang, D. Ramanan and K. Fatahalian, "Online model distillation for efficient video inference", in "Proceedings of the IEEE International Conference on Computer Vision", pp. 3573–3582 (2019).

Netzer, Y., T. Wang, A. Coates, A. Bissacco, B. Wu and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning", URL `http://ufldl.stanford.edu/housenumbers/nips2011`$_h$*ousenumbers.pdf* (2011).

Nguyen, D. V. and J. Choi, "Toward scalable video analytics using compressed-domain features at the edge", Applied Sciences **10**, 18, URL `https://www.mdpi.com/2076-3417/10/18/6391` (2020).

Oliva, A., "Gist of the scene", in "Neurobiology of attention", pp. 251–256 (Elsevier, 2005).

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

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library", Advances in neural information processing systems **32**, 8026–8037 (2019).

Perreault, H., G.-A. Bilodeau, N. Saunier and M. Héritier, "Spotnet: Self-attention multi-task network for object detection", in "2020 17th Conference on Computer and Robot Vision (CRV)", pp. 230–237 (IEEE, 2020).

Powers, D. M., "Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation", (2011).

Prest, A., C. Leistner, J. Civera, C. Schmid and V. Ferrari, "Learning object class detectors from weakly annotated video", in "Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on", pp. 3282–3289 (IEEE, 2012).

Putten, J., F. Van der Sommen and P. With, "Influence of decoder size for binary segmentation tasks in medical imaging", p. 43 (2020).

Qijie, Z. and N. Feng, "pytorch-ssd", `https://github.com/qijiezhao/pytorch-ssd` (2019).

Qiu, J., J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song *et al.*, "Going deeper with embedded fpga platform for convolutional neural network", in "Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays", pp. 26–35 (ACM, 2016).

Rastegari, M., V. Ordonez, J. Redmon and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks", in "European Conference on Computer Vision", pp. 525–542 (Springer, 2016a).

Rastegari, M., V. Ordonez, J. Redmon and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks", (2016b).

Ravì, D., C. Wong, F. Deligianni, M. Berthelot, J. Andreu-Perez, B. Lo and G.-Z. Yang, "Deep learning for health informatics", IEEE journal of biomedical and health informatics **21**, 1, 4–21 (2017).

Redmon, J. and A. Farhadi, "Yolov3: An incremental improvement", arXiv preprint arXiv:1804.02767 (2018).

Ren, S., K. He, R. Girshick and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks", in "Advances in neural information processing systems", pp. 91–99 (2015).

Rezatofighi, H., N. Tsoi, J. Gwak, A. Sadeghian, I. Reid and S. Savarese, "Generalized intersection over union: A metric and a loss for bounding box regression", in "Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition", pp. 658–666 (2019).

Richefeu, J. and A. Manzanera, "A new hybrid differential filter for motion detection", in "Computer Vision and Graphics", pp. 727–732 (Springer, 2006).

Ritchie, J. B., D. A. Tovar and T. A. Carlson, "Emerging Object Representations in the Visual System Predict Reaction Times for Categorization", PLoS Computational Biology **11**, 6, 1–18 (2015).

Romero, A., N. Ballas, S. E. Kahou, A. Chassang, C. Gatta and Y. Bengio, "Fitnets: Hints for thin deep nets", arXiv preprint arXiv:1412.6550 (2014).

Ronneberger, O., P. Fischer and T. Brox, "U-net: Convolutional networks for biomedical image segmentation", (2015).

Sandler, M., A. Howard, M. Zhu, A. Zhmoginov and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks", in "Proceedings of the IEEE conference on computer vision and pattern recognition", pp. 4510–4520 (2018).

Saood, A. and I. Hatem, "Covid-19 lung ct image segmentation using deep learning methods: Unet vs. segnet", (2020).

Sengar, S. S. and S. Mukhopadhyay, "Moving object detection using statistical background subtraction in wavelet compressed domain", Multimedia Tools and Applications **79**, 9, 5919–5940 (2020).

Shen, H., S. Han, M. Philipose and A. Krishnamurthy, "Fast video classification via adaptive cascading of deep models", in "Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition", (2017).

Singh, B., M. Najibi and L. S. Davis, "Sniper: Efficient multi-scale training", arXiv preprint arXiv:1805.09300 (2018).

Sinha, R. S., Y. Wei and S.-H. Hwang, "A survey on lpwa technology: Lora and nb-iot", Ict Express **3**, 1, 14–21 (2017).

Su, J.-C. and S. Maji, "Adapting models to signal degradation using distillation", arXiv preprint arXiv:1604.00433 (2016).

Sultani, W., C. Chen and M. Shah, "Real-world anomaly detection in surveillance videos", in "Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition", pp. 6479–6488 (2018).

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going deeper with convolutions", in "Proceedings of the IEEE conference on computer vision and pattern recognition", pp. 1–9 (2015).

Tan, M., B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile", in "Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition", pp. 2820–2828 (2019).

Tang, J., D. Sun, S. Liu and J.-L. Gaudiot, "Enabling deep learning on iot devices", Computer **50**, 10, 92–96 (2017).

Teerapittayanon, S., B. McDanel and H. Kung, "Branchynet: Fast inference via early exiting from deep neural networks", in "Pattern Recognition (ICPR), 2016 23rd International Conference on", pp. 2464–2469 (IEEE, 2016).

Tzeng, E., J. Hoffman, N. Zhang, K. Saenko and T. Darrell, "Deep domain confusion: Maximizing for domain invariance", arXiv preprint arXiv:1412.3474 (2014).

Umuroglu, Y., N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference", in "Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays", pp. 65–74 (ACM, 2017).

Verelst, T. and T. Tuytelaars, "Dynamic convolutions: Exploiting spatial sparsity for faster inference", (2020).

Viola, P. and M. Jones, "Rapid object detection using a boosted cascade of simple features", in "Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on", vol. 1, pp. I–I (IEEE, 2001).

Wang, H., Y. Xu, B. Ni, L. Zhuang and H. Xu, "Flexible network binarization with layer-wise priority", in "2018 25th IEEE International Conference on Image Processing (ICIP)", pp. 2346–2350 (2018a).

Wang, X., F. Yu, Z.-Y. Dou, T. Darrell and J. E. Gonzalez, "Skipnet: Learning dynamic routing in convolutional networks", in "Proceedings of the European Conference on Computer Vision (ECCV)", pp. 409–424 (2018b).

Webster, M. A., "Visual adaptation", Annual review of vision science **1**, 547–567 (2015).

Wen, L., D. Du, Z. Cai, Z. Lei, M.-C. Chang, H. Qi, J. Lim, M.-H. Yang and S. Lyu, "Ua-detrac: A new benchmark and protocol for multi-object detection and tracking", Computer Vision and Image Understanding **193**, 102907 (2020).

Wen, W., C. Wu, Y. Wang, Y. Chen and H. Li, "Learning structured sparsity in deep neural networks", in "Advances in Neural Information Processing Systems", pp. 2074–2082 (2016).

Wu, J., C. Leng, Y. Wang, Q. Hu and J. Cheng, "Quantized convolutional neural networks for mobile devices", in "Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition", pp. 4820–4828 (2016).

Wu, S., M. Kan, S. Shan and X. Chen, "Hierarchical attention for part-aware face detection", International Journal of Computer Vision **127**, 6, 560–578 (2019a).

Wu, Z., C. Xiong, C.-Y. Ma, R. Socher and L. S. Davis, "Adaframe: Adaptive frame selection for fast video recognition", (2019b).

Xiong, Z., Z. Yao, Y. Ma and X. Wu, "Vikingdet: A real-time person and face detector for surveillance cameras", in "2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)", pp. 1–7 (IEEE, 2019).

Xu, K., M. Qin, F. Sun, Y. Wang, Y.-K. Chen and F. Ren, "Learning in the frequency domain", (2020).

Yang, L., "flexible-yolov5", `https://github.com/yl305237731/flexible-yolov5` (2021).

Ye, X., Z. Lin, H. Li, S. Zheng and Y. Yang, "Active object perceiver: Recognition-guided policy learning for object searching on mobile robots", in "2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)", pp. 6857–6863 (IEEE, 2018).

Zeiler, M. D. and R. Fergus, "Visualizing and understanding convolutional networks", Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **8689 LNCS**, PART 1, 818–833 (2014).

Zhang, L. and K. Ma, "Improve object detection with feature-based knowledge distillation: Towards accurate and efficient detectors", in "International Conference on Learning Representations", (2020).

Zhang, X., X. Zhou, M. Lin and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices", in "Proceedings of the IEEE conference on computer vision and pattern recognition", pp. 6848–6856 (2018).

Zhang, X., J. Zou, K. He and J. Sun, "Accelerating very deep convolutional networks for classification and detection", IEEE transactions on pattern analysis and machine intelligence **38**, 10, 1943–1955 (2016).

Zhao, H., J. Shi, X. Qi, X. Wang and J. Jia, "Pyramid scene parsing network", in "Proceedings of the IEEE conference on computer vision and pattern recognition", pp. 2881–2890 (2017).

Zhou, B., A. Lapedriza, J. Xiao, A. Torralba and A. Oliva, "Learning deep features for scene recognition using places database", in "Advances in neural information processing systems", pp. 487–495 (2014).

Zhou, H.-Y., B.-B. Gao and J. Wu, "Adaptive feeding: Achieving fast and accurate detections by adaptively combining object detectors", in "Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition", pp. 3505–3513 (2017).

Zhou, S., Y. Wu, Z. Ni, X. Zhou, H. Wen and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients", arXiv preprint arXiv:1606.06160 (2016).

Zhu, X., Y. Wang, J. Dai, L. Yuan and Y. Wei, "Flow-guided feature aggregation for video object detection", in "Proceedings of the IEEE International Conference on Computer Vision", pp. 408–417 (2017).