Adaptive Compression for Online Computer Vision: an Edge Reinforcement Learning Approach

ZHAOLIANG HE, Department of Computer Science and Techonology, Tsinghua University, and Peng Cheng Laboratory
HONGSHAN LI, Tsinghua-Berkeley Shenzhen Institute, Tsinghua University
ZHI WANG, Tsinghua Shenzhen International Graduate School, Tsinghua University, and Peng Cheng Laboratory
SHUTAO XIA, Tsinghua Shenzhen International Graduate School, Tsinghua University
WENWU ZHU, Department of Computer Science and Techonology, Tsinghua University

With the growth of computer vision-based applications, an explosive amount of images have been uploaded to cloud servers that host such online computer vision algorithms, usually in the form of deep learning models. JPEG has been used as the *de facto* compression and encapsulation method for images. However, standard JPEG configuration does not always perform well for compressing images that are to be processed by a deep learning model, e.g., the standard quality level of JPEG leads to 50% of size overhead (compared with the best quality level selection) on ImageNet under the same inference accuracy in popular computer vision models (e.g., InceptionNet and ResNet). Knowing this, designing a better JPEG configuration for online computer vision-based services is still extremely challenging: 1) Cloud-based computer vision models are usually a black box to end-users; thus, it is challenging to design JPEG configuration without knowing their model structures. 2) The "optimal" JPEG configuration is not fixed; instead, it is determined by confounding factors, including the characteristics of the input images and the model, the expected accuracy and image size, etc. In this paper, we propose a reinforcement learning (RL)-based adaptive JPEG configuration framework, AdaCompress. In particular, we design an edge (i.e., user-side) reinforcement learning agent that learns the optimal compression quality level to achieve an expected inference accuracy and upload image size, only from the online inference results, without knowing details of the model structures. Furthermore, we design an *explore-exploit* mechanism to let the framework fast switch an agent when it detects a performance degradation, mainly due to the input change (e.g., images captured across daytime and night). Our evaluation experiments using real-world online computer vision-based APIs from Amazon Rekognition, Face++, and Baidu Vision, show that our approach outperforms existing baselines by reducing the size of images by 1/2 – 1/3 while the overall classification accuracy only decreases slightly; Meanwhile, AdaCompress adaptively re-trains or re-loads the RL agent promptly to maintain the performance.

CCS Concepts: • **Networks** → *Network components*; • **Computer systems organization** → *Real-time systems.*

Additional Key Words and Phrases: Edge Computing, Reinforcement Learning, Adaptive Compression, Machine Learning Service

1 INTRODUCTION

With the great success of deep learning in computer vision, this decade has witnessed an explosion of deep learning-based computer vision-based applications. Because of the enormous computational resource consumption for deep learning applications (e.g., inferring an image on VGG19 [49] requires 20 GFLOPs of GPU resource), in today's online computer vision-based applications, users

The preliminary version of this paper was published in ACM Multimedia 2019 [33].
Authors' addresses: Zhaoliang He, Department of Computer Science and Techonology, Tsinghua University, and Peng Cheng Laboratory, hezl19@mails.tsinghua.edu.cn; Hongshan Li, Tsinghua-Berkeley Shenzhen Institute, Tsinghua University, lhs17@mails.tsinghua.edu.cn; Zhi Wang, Tsinghua Shenzhen International Graduate School, Tsinghua University, and Peng Cheng Laboratory, wangzhi@sz.tsinghua.edu.cn; Shutao Xia, Tsinghua Shenzhen International Graduate School, Tsinghua University, xiast@sz.tsinghua.edu.cn; Wenwu Zhu, Department of Computer Science and Techonology, Tsinghua University, wwzhu@tsinghua.edu.cn.

usually have to upload the input images to the central cloud service providers (e.g., SenseTime, Baidu Vision and Google Vision, etc.), leading to a significant upload traffic load.

To reduce the upload traffic load, one should compress the image before uploading it. Though JPEG has been used as the *de facto* image compression and encapsulation method, its performance for the deep computer vision models is not satisfactory. Liu et al. [36] showed that by re-designing the quantization table in the default JPEG configuration, one can compress an image to a smaller version while maintaining the comparable inference accuracy for a deep computer vision model. However, such quantization solutions usually assume the inference model is fixed.

We then raise an intuitive question: to make it practically useful, can we select the JPEG configuration adaptively for different online computer vision-based services, without any prior knowledge of the original model? In this paper, we propose a reinforcement learning-based framework to select JPEG configurations adaptively. In our solution, we tackle the following design challenges.

- *Lack of information about the cloud-based computer vision models.* Previous studies [21, 36, 56], generally assume that the details of the computer vision models are available so that they can adjust the JPEG configuration according to the model structure, e.g., one can train a model to determine the JPEG configuration by plugging the original computer vision model into it. However, the structural details of online computer vision models are usually proprietary and not open to the users.
- *Different cloud-based computer vision models need different JPEG configurations.* As an adaptive JPEG configuration solution, we target to provide a solution that is adaptive to different online computer vision-based services, i.e., it can *generate* JPEG configuration for different models. However, today's cloud-based computer vision algorithms, based on deep and convolutional computations, are quite hard to understand. The same compression quality level could lead to a different accuracy performance. Some examples are shown in Figure 1: picture 1a and 1b, 2a and 2b are visually similar for human beings, but the deep learning model gives different inference results, only because they are compressed at different quality levels. And such a relationship is not apparent, e.g., picture 3b is highly compressed and looks destroyed comparing to picture 3a, but the deep learning model can still recognize it. This phenomenon is also presented in [11] and commonly seen in adversarial neural network researches [15, 59].
- *Lack of well-labeled training data.* In our problem, one is not provided the well-labeled data on which image should be compressed at which quality level, as in conventional supervised deep learning tasks. In practice, such an image compression module is usually utilized in an online manner, and the solution has to learn from the images it uploads automatically.

To address the above challenges, we present a reinforcement learning-based solution, called AdaCompress[1], to choose a proper compression quality level for an image to a computer vision model on the cloud-end, in an online manner. This paper is an extension of our earlier conference paper [33], with the following contributions:

- We design an interactive training environment that can be applied to different online computer vision-based services. We propose a Deep Q-learning Network-based [37] agent to evaluate and predict the performance of a compression quality level on an input image. In real-world application scenarios, this agent can be highly efficient to run on today's edge infrastructures (e.g., Google edge TPU [20], Huawei Atlas 500 edge station [28]).
- We build a reinforcement learning-based framework to train the agent in the above environment. The agent can learn to choose a proper compression quality level for an input image after iteratively interacting with the environment by feeding the carefully designed

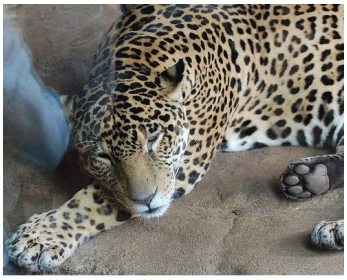
[1]We open-sourced AdaCompress that works with online computer vision-based APIs at https://github.com/hosea1008/AdaCompress

(1a) Q=75
Face++ prediction = [“donut”]

(1b) Q=55
Face++ prediction = [“biscuit”]

(2a) Q=75
Baidu prediction = [“chameleon”]

(2b) Q=55
Baidu prediction = [“electric fan”]



(3a) Q=75
Baidu prediction = [“leopard”]

(3b) Q=5
Baidu prediction = [“leopard”]

Figure 1. The prediction of a deep learning model is not completely related to the input image’s quality, making it difficult to use a fixed compression quality for all images. For image 1a, 1b and 2a, 2b, minor changes cause different predictions though they are visually similar; for image 3a and 3b, the cloud-based model still output correct label from a severely compressed image though they look very different

reward that considers both accuracy and data size. We further propose an *explore-exploit* mechanism to let the agent switch between “sceneries”. In particular, after deploying the agent, an *inference-estimation-querying-retraining* solution is designed to switch the RL agent intelligently once the scenery changes and the existing running agent cannot guarantee the original accuracy performance.

- In this journal extension, we provide more analysis and insights on our design. We notice that the Deep Q-learning Network-based agent’s behaviors are various for different input image “sceneries” and backend cloud services. By analyzing the agent’s behaviors using

Grad-Cam [48], we provide the reasons that the agent chooses a specific compression quality level. We reveal that images containing large smooth areas are more sensitive to compression, while images with complex textures are more robust to compression for computer vision models.
- We evaluate our system on representative cloud-based deep learning services, including Amazon Rekognition [2], Face++ [16] and Baidu Vision [5]. We show that our design can reduce the upload traffic load by up to 1/2 while maintaining comparable overall accuracy. Compared to baseline DeepN-JPEG [36], the overall accuracy of AdaCompress is 8% higher when they have similar compressed image size.

The rest of this paper is organized as follows. We discuss related works in Section 2. We present our framework and detailed design in Section 3. We present our solution's performance in Section 4 and conclude the paper in Section 5.

2 RELATED WORKS

As online computer vision-based services have become the norm for today's applications [1, 29], many studies have been devoted to improving the cloud-based model execution, including model compression and data compression.

2.1 Model Compression

Though the accurate term is still for the community to debate, we use "model compression" to represent the studies on *compressing* and *moving* the deep learning models close to users. Many studies tried to compress the deep learning models and deploy them *locally* [3, 4, 19, 22, 23, 30], i.e., running an alternative "smaller version" of a computer vision model at the user-side, to avoid the image upload so that to improve the inference efficiency. Other studies proposed to run a part of a deep learning model *locally* [14, 26, 31, 34], by decoupling the deep learning model into different parts, e.g., based on the layers in the deep learning model, so that a part of the inference is done *locally* to save some execution time. However, these solutions usually need to re-train the model using the original dataset of the model, which is not practical for today's online computer vision-based services that are merely a black box to end-users, e.g., in the form of a RESTful API.

2.2 Data Compression

Data compression solutions study how to compress the original data (e.g., a video or image) to be inferred by the cloud-based deep learning model so that less traffic is used to upload the data to improve inference speed. In recent years, researchers found that conventional human visually optimized-based data compression solutions (e.g., JPEG [57], WebP [7] and JPEG2000 [40], etc.) and some recent neural network-based compression solutions [43, 53–55] are not usually applicable to deep learning vision models. Liu et al. [36] revealed that the conventional JPEG image compression framework is designed for the Human-Visual System, which is not suitable for the deep neural network, leading to the computer vision model's inference performance degradation. Dodge et al. [13] further discovered that besides the JPEG compression, four types of quality distortions (blur, noise, contrast, and the JPEG2000 compression [40]) can also affect the inference performance of the deep learning models. Delac et al. [11] observed that, in some cases, a high compression quality level does not always reduce the model inference accuracy.

Based on these insights, Robert et al. [56] tried to train the deep neural network from the compressed representations of an auto-encoder. Chao et al. [10] proposed using variable quantization, which is supported by the JPEG standard extension syntax [12] to compress the macroblocks in

images. Furthermore, they [9] designed a quantization table based on the observed impact of scale-space processing on the discrete cosine transform (DCT) basis functions for JPEG images, achieving similar inference performance while reducing the image size overhead effectively. Liu et al. [36] proposed DeepN-JPEG that re-designs the quantization table by linking statistical information of defined features and defined quantization values so that the compressed image size is reduced for deep learning models. Chamain et al. [8] proposed a joint optimization of image classification network coupled with the image quantization, achieving image size reduction of JPGE2000 [40] encoded images. Recently, Lionel et al. [21] presented a new type of neural network that infers directly from the discrete cosine transform coefficients in the middle of the JPEG codec. Baluja et al. [6] proposed task-specific compression that compresses images based on the end-use of images.

However, such proposals all need one to understand the characteristics of the cloud-based deep learning model and have access to the original training dataset, *generating* the appropriate compression schemes. To the best of our knowledge, we are the first to propose an adaptive compression configuration solution that learns the optimal compression quality level to achieve an expected inference accuracy and upload image size, only from the online inference results, without knowing details of the model structures.

In this journal extension, we improve the previous *inference-estimate-retrain* mechanism to cut down the upload image size overhead effectively, add DeepN-JPEG comparative experiment, and amend the manuscript significantly. Especially in the DeepN-JPEG comparative experiment, since Liu et al. [36] evaluated the DeepN-JPEG framework on ImageNet by using four state-of-the-art DNN models (AlexNet [32], VGG [50], GoogLeNet [52] and ResNet [25]) on local edge devices, which are different from online computer vision-based services, for comparison purpose, we implement the DeepN-JPEG framework according to their paper and evaluate the size reduction and accuracy performance using ImageNet and the metrics in Subsection 4.3 on three cloud-based deep learning services (Amazon Rekognition, Face++ and Baidu Vision).

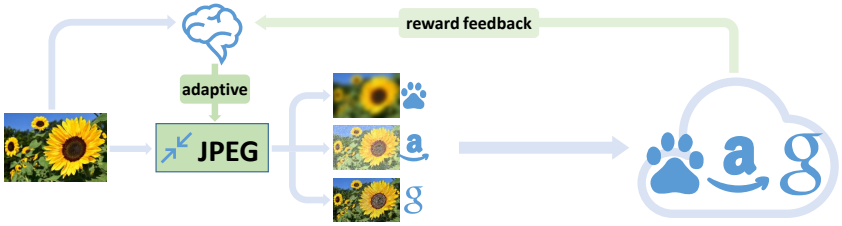
3 DETAILED DESIGN

A brief framework of AdaCompress is shown in Figure 2. Briefly, it is a reinforcement learning-based system to train an agent to choose a proper compression quality level c for one image to be compressed by JPEG. We discuss the formulation, agent design, reinforcement learning-based framework, reward feedback, *inference-estimation-querying-retraining* mechanism, and insight of RL agent's behaviors separately in the following subsections. We provide experimental details of all the hyperparameters in Section 4.

3.1 Problem Formulation

Without loss of generality, we denote the cloud-based deep learning service as $\vec{y}_i = M(x_i)$ that provides a predicted result list $\vec{y}_i$ for each input image x_i. It has a baseline output $\vec{y}_{\text{ref}} = M(x_{\text{ref}})$ for each reference input image $x \in X_{\text{ref}}$. We use this $\vec{y}_{\text{ref}}$ as the ground truth label. For each image x_c compressed at compression quality level c, the output $\vec{y}_c = M(x_c)$. Therefore, we have an accuracy metric $\mathcal{A}_c$ by comparing $\vec{y}_{\text{ref}}$ and $\vec{y}_c$. In general, we use the top-5 accuracy as the following $\mathcal{A}$, the same as the classification metric of ILSVRC2012 [38].

(a) Conventional solution: fixed user-defined compression quality level



(b) AdaCompress solution: input image and model aware compression

Figure 2. Comparing to the conventional solution, our solution can update the compression strategy based on the backend model feedback

$$\mathcal{A} = \frac{1}{k}\sum_{k}\max_{j} d(l_j, g_k) \tag{1}$$

$$l_j \in \vec{y}_c, \quad j = 1, ..., 5 \tag{2}$$

$$g_k \in \vec{y}_{\text{ref}}, \quad k = 1, ..., \text{length}(\vec{y}_{\text{ref}}) \tag{3}$$

$$d(x, y) = 1 \text{ if } x = y \text{ else } 0 \tag{4}$$

Where $j = 1, ..., 5$ indicates the prediction labels at the top-5 score, meaning that if any one of the top-5 predicted labels matches the ground truth label $\vec{y}_{\text{ref}}$, it would be regarded as a correct prediction. In general, we cannot get the cloud-based deep learning model's in-layer details (e.g., softmax probabilities) for a cloud-based deep learning service. Therefore we use a binary hard label $d(x, y) \in \{0, 1\}$ to evaluate the accuracy.

We also denote JPEG input images as $f_{ic} = J(x_i, c)$ that for an input image x_i and a given compression quality level c, and it outputs a compressed file f_{ic} at the size of $\hat{s}_{ic}$. For a reference compression quality level c_{ref}, the compressed file size is $\hat{s}_{\text{ref}}$. Besides, the input image from a specific scenery usually belongs to a particular contextual group [24]. For example, in the daytime, the input images are expected to have a bright background, while nighttime images are usually gray-scaled thermal images. Therefore, the agent in one scenery does not need to know all the contextual features in all "sceneries". We formulate this as contextual group $\mathcal{X}$.

Initially, the agent tries different compression quality levels $c_{\min} < c < c_{\max}, c \in \mathbb{N}$ to obtain compressed image x_c from input image x. To obtain cloud-end recognition results $\{\vec{y}_{\text{ref}}, \vec{y}_c\}$, the agent uploads the compressed image x_c and the reference image x_{ref} to the cloud-end. Comparing the two uploaded instances $\{x_{\text{ref}}, x_c\}$ and cloud-end recognition results $\{\vec{y}_{\text{ref}}, \vec{y}_c\}$, the agent obtains the reference file size $\hat{s}_{\text{ref}}$ and compressed file size $\hat{s}_c$, and computes the file compression ratio $\Delta s = \frac{\hat{s}_c}{\hat{s}_{\text{ref}}}$ and accuracy metric $\mathcal{A}_c$.

3.2 RL Agent Design

The RL agent is expected to give a proper compression quality level c for minimizing the file size $\hat{s}_c$ while keeping the accuracy $\mathcal{A}$. In our design, each compression is treated as a complete RL task that only has one step, and the agent performs the action only once to accomplish the task. The agent is able to learn across consecutive tasks, because the input sequence (i.e., consecutive images likely to be captured by the same camera) shares similar contextual characteristics. The similar design has also been used in previous studies, including RL-based cache strategy [45, 58, 60]. For the RL agent, the input features are continuous numerical vectors, and the expected output is discrete compression quality level c. Therefore we can use the Deep Q-learning Network as the RL agent. But the naive Deep Q-learning Network can not work well in this task because of the following challenges:

- The state space of reinforcement learning is too large. To preserve enough details, we have to add many layers and nodes to the neural network, making the RL agent extremely difficult to converge.
- It takes a long time to train one step in a large inference neural network, making the training process too time-consuming.
- The RL agent starts training from random trials and learns afterward from the reward feedback. When training from a randomly initialized neural network, the reward feedback is very sparse, making it difficult for the agent to learn.

To address these challenges, we use the early layers of a pre-trained neural network to extract the structural information of an input image. This is a commonly used strategy in training a deep neural network [18, 41]. Therefore instead of training a RL agent directly from the input image, we use a pre-trained small neural network to extract the features from the input image to reduce the input dimension and accelerate the training procedure. In this work, we use the early convolution layers of MobileNetV2 [46] as the image feature extractor $\mathcal{E}(\cdot)$ for its efficiency in image classification. The Deep Q-learning Network ϕ is connected to the feature extractor's last convolution layer. We update the RL agent's policy by changing the parameters of the Deep Q-learning Network ϕ while fixing the feature extractor $\mathcal{E}$.

3.3 Reinforcement Learning-based Framework

In a specific scenery where the user input image x belongs to the contextual group $\mathcal{X}$, we define the contextual group $\mathcal{X}$, along with the backend cloud model M, as the *emulator environment* $\{\mathcal{X}, M\}$ of the reinforcement learning problem.

We formulate the feature extractor's output $s = \mathcal{E}(J(\mathcal{X}, c))$ as *states* and the compression quality level c as discrete *actions*. In our system, to accelerate training, we define 10 discrete actions to indicate 10 compression quality levels of JPEG ranging from $5, 15, ..., 95$. We denote the *action-value function* as $Q(\mathcal{E}(f_t), c; \theta)$ and the optimal compression quality level at time t as $c_t = \text{argmax}_c Q(\mathcal{E}(f_t), c; \theta)$ where θ indicates the parameters of the Deep Q-learning Network ϕ. In such reinforcement learning formulation, the training phase is to minimize the loss function $L_i(\theta_i) = \mathbb{E}_{s,c\sim\rho(\cdot)}\left[\left(y_i - Q(s, c; \theta_i)\right)^2\right]$ that changes at each iteration i where state $s = \mathcal{E}(f_i)$ and target $y_i = \mathbb{E}_{s'\sim\{\mathcal{X},M\}}\left[r + \gamma \max_{c'} Q(s', c'; \theta_{i-1}) \mid s, c\right]$. Especially, r is the reward feedback, and $\rho(s, c)$ is a probability distribution over state s and the compression quality level c [37]. When minimizing the distance of *action-value function*'s output $Q(\cdot)$ and target y_i, the *action-value function* $Q(\cdot)$ outputs a more accurate estimation of an action.

Different from conventional reinforcement learning, the interactions between the agent and environment are infinite, i.e., there is no signal from the environment telling that an episode has

Algorithm 1 Training RL agent ϕ in environment $\{\mathcal{X}, M\}$

Initialize replay memory buffer $\mathcal{D}$ to capacity N
Initialize *action-value function* Q with random weights θ
Initialize state $s_1 \leftarrow \mathcal{E}\big(J(x_1, c_{ref})\big), x_1 \in \mathcal{X}$
for $t \in 1, 2, \cdots, K$ **do**
  1) Exploration
  With probability ϵ:
    $c_t \leftarrow$ a random valid value
  Otherwise:
    $c_t \leftarrow \text{argmax}_c Q(\mathcal{E}(f_t), c; \theta)$

  2) Reward calcuation
  Compress image x_t at quality c_t to upload
  Receive $(\vec{y}_{\text{ref}}, \vec{y}_c)$ from the cloud service
  Compute reward $r \leftarrow R(\Delta s, \mathcal{A}_c)$ according to 3.4 Reward Feedback Design

  3) Gradient descent
  Obtain next image x_{t+1}
  Generate next state $s_{t+1} \leftarrow \mathcal{E}\big(J(x_{t+1}, c_{ref})\big)$
  $\mathcal{D} \leftarrow \mathcal{D} \bigcup \{(s_t, c_t, r_t, s_{t+1}, \mathcal{A}_t)\}$
  if $t \mod T = 0$ and $t \geq T_{\text{start}}$ **then**
    Sample a randomly mini-batch of transitions $(s_j, c_j, r_j, s_{j+1}, \mathcal{A}_j)$ from memory buffer $\mathcal{D}$
    $y_j \leftarrow r_j + \gamma \max_{c'} Q(s_{j+1}, c'; \theta)$
    Compute decay exploration rate $\epsilon \leftarrow \begin{cases} \mu_{\text{dec}} \cdot \epsilon & \text{if } \mu_{\text{dec}} \cdot \epsilon > \epsilon_{\text{min}} \\ \epsilon_{\text{min}} & \text{if } \mu_{\text{dec}} \cdot \epsilon \leq \epsilon_{\text{min}} \end{cases}$
    Perform a gradient descent step on $\big(y_j - Q(s_j, c_j; \theta)\big)^2$ according to [37]
  end if
end for

finished. Therefore, we train the RL agent intermittently at a manual interval of T after the condition $t \geq T_{\text{start}}$ guaranteeing that there are enough transitions in the memory buffer $\mathcal{D}$. In the training phase, the RL agent firstly takes some random trials to observe the environment's reaction and decreases the randomness when training afterward. In iteration t, we leverage the feature extractor $\mathcal{E}$ to obtain state $s_t = \mathcal{E}(J(x_t, c_{ref}))$. The RL agent ϕ generates a compresion quality level c_t. The framework compresses image x_t at quality c_t to upload and obtains reward r_t. The framework obtains next image x_{t+1} and generates next state $s_{t+1} = \mathcal{E}\big(J(x_{t+1}, c_{ref})\big)$. The framework stores transition $(s_t, c_t, r_t, s_{t+1}, \mathcal{A}_t)$ in a memory buffer $\mathcal{D}$. Especially, (s_t, c_t, r_t, s_{t+1}) is uesd to compute the loss function, and $(r_t, \mathcal{A}_t)$ is used in 3.5 Inference-Estimation-Querying-Retraining Mechanism. All transitions are saved into a memory buffer $\mathcal{D}$, and the agent learns to optimize its *action* by minimizing the loss function L on a mini-batch from $\mathcal{D}$. The training procedure would converge when the agent's randomness keeps decaying. Finally, the agent's action is based on its historical "optimal" experiences. The training procedure is presented in Algorithm 1.

3.4 Reward Feedback Design

In our solution, the agent is trained by the reward feedback from the environment $\{\mathcal{X}, M\}$. In the above formulation, we define compression rate $\Delta s = \frac{\hat{s}_c}{\hat{s}_{\text{ref}}}$ and accuracy metric $\mathcal{A}_c$ at compression

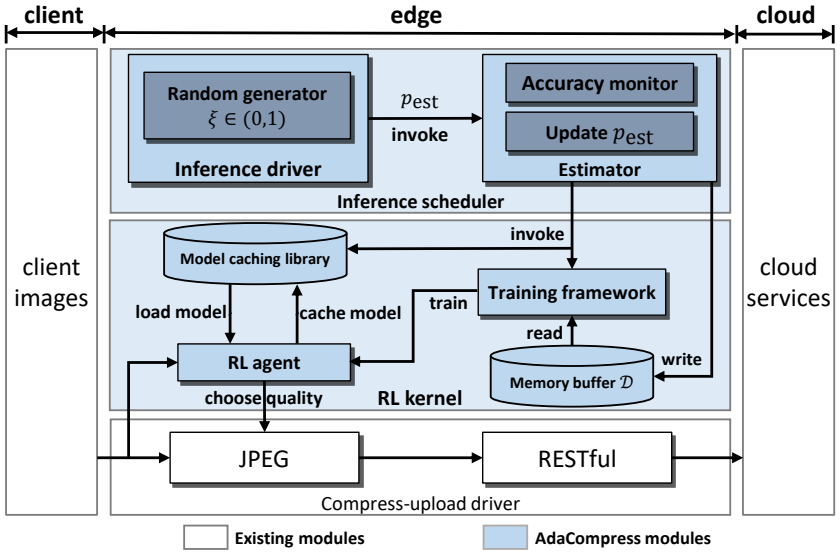


Figure 3. Diagram of AdaCompress architecture

quality level c. Basically, we want the agent to choose a proper compression quality level for minimizing the upload image size while remaining acceptable accuracy. Therefore the overall reward r should be positively correlated with the accuracy $\mathcal{A}$ while negatively with the compression ratio Δs. We introduce two linear factors α and β to form a linear combination $r = \alpha\mathcal{A} - \Delta s + \beta$ as the *reward function* $R(\Delta s, \mathcal{A})$.

3.5 Inference-Estimation-Querying-Retraining Mechanism

As a running system, we introduce an *inference-estimation-querying-retraining* mechanism to cope with the scenery change in the inference phase, building a system with different components to inferring, capturing the scenery change, then either re-loading or re-training the RL agent based on the performance of the cached model. The overall system diagram is illustrated in Figure 3.

We build up the memory buffer $\mathcal{D}$ and reinforcement learning training kernel based on the compression and upload driver. When the RL training kernel is called, it would load transitions from the memory buffer $\mathcal{D}$ to train the compression quality level predictor ϕ. When the system is deployed, the pre-trained RL agent ϕ guides the compression driver to compress the input image at an adaptive compression quality level c, then uploads the compressed image to cloud-end.

After the AdaCompress is deployed, the input image scenery context $\mathcal{X}$ may change (e.g., day to night, sunny to rainy). When the scenery changes, the older RL agent's compression selection strategy may not be suitable anymore, causing the overall accuracy to decrease. To cope with the scenery change, AdaCompress invokes an estimator with a probability p_{est}. AdaCompress does this by generating a random value $\xi \in (0, 1)$ and comparing it to p_{est}. If $\xi \leq p_{\text{est}}$, the estimator would be invoked. AdaCompress would upload the reference image x_{ref} along with the compressed image x_i to obtain $\vec{y}_{\text{ref}}$ and $\vec{y}_i$, calculate $\mathcal{A}_i$ and save the transition $(s_i, c_i, r_i, s_{i+1}, \mathcal{A}_i)$ to the memory buffer $\mathcal{D}$. The estimator also compares the recent n steps' average accuracy $\bar{\mathcal{A}}_n$ and the initial accuracy threshold $\mathcal{A}_0$. Once the recent average accuracy is lower than the initial accuracy threshold, the estimator would query a RL agent model to replace the current agent model and test the performance of the loaded model. To test the loaded model's performance, the estimator computes the average

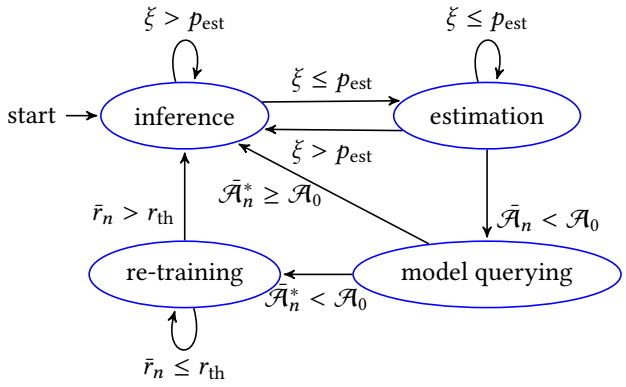


Figure 4. State switching policy

accuracy $\bar{\mathcal{A}}_n^*$. If $\bar{\mathcal{A}}_n^*$ is still lower than the accuracy threshold, the estimator would invoke the RL training kernel to re-train the agent. Once the estimator discovers that the trained reward is higher than the reward threshold, it would stop the training kernel, cache the trained RL agent, and switch to the normal inference state.

Since the reference image x_{ref} and the compressed image x_i are both needed in the re-training phase, causing a large upload image size overhead, especially when the scenery changes frequently (e.g., day to night, then night to day). To avoid unnecessary upload traffic load in the re-training phase, we build up the model caching library to cache the trained RL agent models. When capturing the scenery change, we firstly query a pre-trained model from the model caching library rather than re-training from scratch.

The *inference-estimation-querying-retraining* mechanism has four states, including inference, estimation, model querying and re-training. AdaCompress switches to these states adaptively. The state switching policy is shown as Figure 4.

3.5.1 Inference: Most of the time, AdaCompress runs in this state. In this state, only the compressed images are uploaded to the cloud-end to achieve less upload image size overhead. To keep a stable accuracy performance even the input image scenery changes, the agent would occasionally switch to the estimation state with probability p_{est} since the estimator uploads the reference image to maintain inference accuracy, or remain in the inference state with probability $1 - p_{\text{est}}$.

3.5.2 Estimation: In this state, the reference image x_{ref} and compressed image x_i are uploaded to the cloud-end simultaneously to obtain $\vec{y}_{\text{ref}}$ and $\vec{y}_i$ which are used to calculate $\mathcal{A}_i$. In each epoch i, the transition $(s_i, c_i, r_i, s_{i+1}, \mathcal{A}_i)$ is logged in the memory buffer $\mathcal{D}$. When the average accuracy $\bar{\mathcal{A}}_n$ of the latest n steps is higher than the accuracy threshold $\mathcal{A}_0$, the agent would stay in the estimation state with probability p_{est} or switch to the inference state with probability $1 - p_{\text{est}}$. Once the average accuracy $\bar{\mathcal{A}}_n$ is lower than the initial accuracy threshold $\mathcal{A}_0$, indicating that the current agent is no more suitable for the current input image scenery, AdaCompress would turn into the model querying state and re-load a new RL agent model from the model caching library.

Therefore, the estimation probability of p_{est} is vital to the whole system. On the one hand, the estimator should be invoked occasionally to estimate the current agent's accuracy for capturing the scenery change on time. On the other hand, the estimator uploads the reference image along with the compressed image. Therefore, the upload image size overhead is greater than the conventional benchmark solution, causing a high upload traffic load.

To achieve trade-off between the risk of the scenery change and the objective of reducing upload traffic load, we design an accuracy-aware dynamic solution. We first define that after running for N steps, the average accuracy of recent n steps is:

$$\bar{\mathcal{A}}_n = \begin{cases} \frac{1}{n}\sum_{i=N-n+1}^{N}\mathcal{A}_i & \text{if } N \geq n \\ \frac{1}{N}\sum_{i=1}^{N}\mathcal{A}_i & \text{if } N < n \end{cases}$$

With this definition, the change of p_{est} should be negatively correlated with the gradient of $\mathcal{A}$, meaning that when the recent accuracy decreases, the estimation probability p_{est} would increase. We define that $p'_{\text{est}} = p_{\text{est}} + \omega\nabla\bar{\mathcal{A}}$ where ω is a negative scaling factor. With this recursive formula, we have the general term of $p_{\text{est}} = p_0 + \omega\sum_{i=1}^{N}\nabla\bar{\mathcal{A}}_i$ where p_0 is an initial estimation probability.

3.5.3 Model Querying: The model querying state is designed to cope with the scenery change before re-training. It re-loads a new RL agent model from the model caching library and tests whether the loaded agent model is suitable for the current scenery by re-calculating the average accuracy. If the new average accuracy $\bar{\mathcal{A}}^*_n$ is higher than the accuracy threshold $\mathcal{A}_0$, indicating that the loaded agent model is suitable for the current scenery, AdaCompress would switch to the normal inference state and use the loaded agent model to infer. Otherwise, AdaCompress would switch to the re-training state and invoke the RL training kernel to re-train a new RL agent for the current scenery.

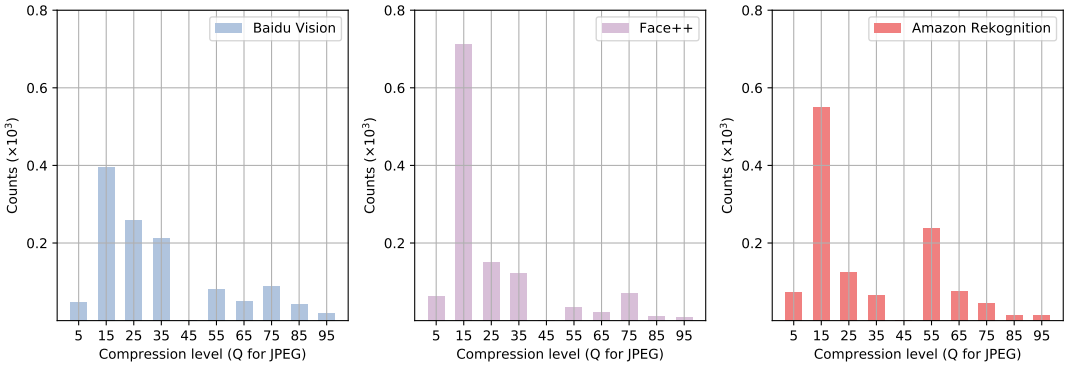
In this way, AdaCompress is capable to cope with the scenery change in a cached "memory" manner, avoiding re-training the agent model at every scenery change to cut down the upload image size overhead. However, the agent caching strategy would cause a little storage overhead on local edge devices.

3.5.4 Re-training: This state is to adapt the agent to the current input image scenery by re-training it with the memory buffer $\mathcal{D}$, which is similar to the training procedure. The re-training phase has finished upon the recent n steps' average reward $\bar{r}_n$ is higher than the user-defined threshold r_{th}. And when the re-training procedure finishes, the memory buffer $\mathcal{D}$ would be flushed, preparing to save new transitions for the re-training of a next scenery change. The trained RL agent model would be cached in the model caching library and be used to switch to the inference state.

3.6 Insight of RL Agent's Behaviors

In the inference phase, the pre-trained RL agent chooses a proper compression quality level according to the input image's features and the backend service. The reference image is not uploaded to the cloud-end anymore; only the compressed image is uploaded. Therefore, the upload traffic load is reduced. We notice that the RL agent's behaviors are various for different input image "sceneries" and backend cloud services. Therefore we try to make further investigations by plotting the RL agent's "attention map" (i.e., visual explanations why the agent chooses a specific compression quality level).

3.6.1 Compression Quality Level Choice Variation: As shown in Figure 5, we find that in different cloud-based application environments, the agent's chosen compression quality levels can be quite different. We compute the *mean* and *standard deviation* of compression quality levels. The mean is 33.4, 24.7 and 31.6 on Baidu Vison, Face++ and Amazon Rekognition, respectively. For Face++, the *mean* of compression quality levels is lower than that for Baidu Vision. The standard deviation is 22.77, 19.19 and 22.46 on Baidu Vison, Face++ and Amazon Rekognition, respectively. The "optimal" compression strategies are different for different backend cloud services. This variation is caused by the interaction between the agent and the backend cloud model in the training phase. Since the



(a) Baidu Vision (b) Face++ (c) Amazon Rekognition

Figure 5. Histogram of RL agent's best compression quality level selection for different cloud services

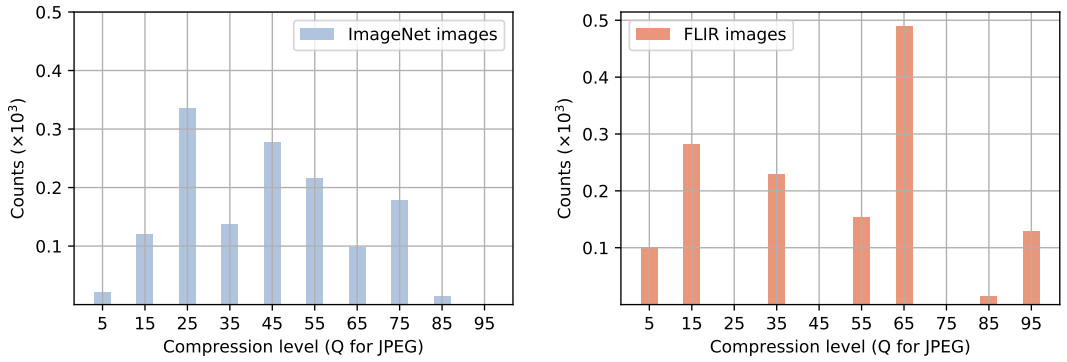
agent's training procedure is based on a specific backend cloud model M_1, for another backend cloud model M_2, the interaction between the agent and M_2 is quite different. Therefore the agent's best compression quality level selection presents variation for different backend cloud models.

Moreover, in our experiment, the agent presents different behaviors when the input image changes from one dataset to another. Figure 6 shows the agent's choices for the same backend cloud model (Baidu Vision) but different image datasets. We prepare two datasets indicating two contextual "sceneries". We randomly sample images from ImageNet [44] whose images are mostly taken in the daytime, to act as a daytime scenery, and randomly select nighttime images from the FLIR Thermal Dataset [17] to form another dataset to act as a nighttime scenery. The histogram shown in Figure 6 points out that the distribution of compression quality levels is different on ImageNet and FLIR Thermal Dataset. For ImageNet and FLIR, the mean is 43.3 and 47.5, respectively; and the standard deviation is 19.52 and 26.56, respectively. To maintain high accuracy when the input image's contextual group $\mathcal{X}$ changes, the agent's compression quality level selection should change as well. This phenomenon presents that the agent can adaptively choose a proper compression quality level based on the input image's features.

We can also find that Figure 5(a) and Figure 6(a) is different, even if we use the same cloud service (Baidu Vision) and image dataset (ImageNet). When we use the same cloud service at different times, the cloud service commonly invokes different backend cloud models. Therefore the interaction between the agent and the backend cloud model is different. This variation presents that the agent can adaptively make a proper compression strategy in different complex environments, indicating our solution's generality and practicality.

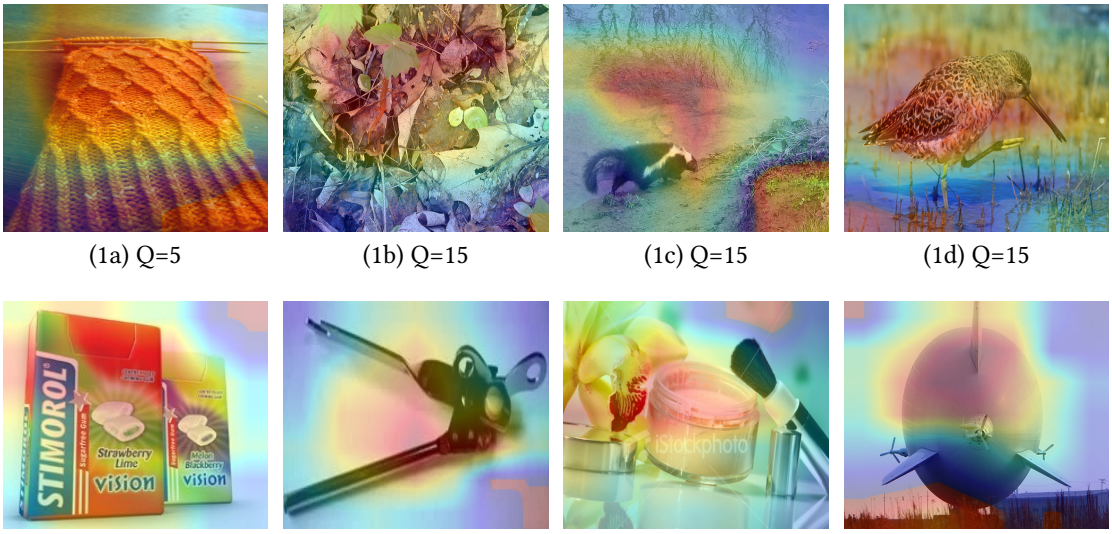
3.6.2 Attention Map Variation: To make insight investigations, we plot the importance map of a chosen compression quality level. We leverage a conventional visualization algorithm, Grad-Cam, to observe the Deep Q-Learning Network-based agent's interest when choosing compression quality levels. Grad-Cam is a widely used effective solution to present the importance map of a deep neural network by calculating the gradients of each target concept and backtracking to the final convolution layer. In this work, we plot the RL agent's attention map by Grad-Cam in Figure 7.

In our investigations, we find that in different environments $\{\mathcal{X}, M\}$, the RL agent selects compression quality levels based on the visual textures of different regions in the image. As shown in Figure 7, picture 1a – 1d are some pictures which the agent chooses to compress aggressively. The agent selects low compression quality levels based on the complex texture of the images. On the



(a) ImageNet images (b) FLIR images

Figure 6. Histogram of RL agent's best compression quality level selection for different scenery image inputs



(1a) Q=5 (1b) Q=15 (1c) Q=15 (1d) Q=15

(2a) Q=85 (2b) Q=85 (2c) Q=75 (2d) Q=75

Figure 7. Visualization of the importance map for the RL agent to choose a compression quality level

contrary, for pictures 2a – 2d, the agent chooses relatively higher compression quality levels to preserve more details since its interest falls on some smooth regions. Especially for 1a and 2a, for picture 1a, the agent chooses a low compression quality level based on the rough central region though there are smooth regions around it, for picture 2a, the agent chooses a relatively higher compression quality level based on the surrounding smooth region rather than the central region.

4 EVALUATION

In this section, we present AdaCompress's behaviors and effectiveness by some real-world experiments.

Table 1. Experiment parameter

Notation	Value	Notation	Value
c_{ref}	75	K	1000
ϵ_{min}	0.02	p_0	0.2
γ	0.95	ω	-3
μ_{dec}	0.99	T	5
r_{th}	0.45	n	10
$\mathcal{A}_0$	0.8	T_{start}	128
α	1	β	0

4.1 Experiment Setup

We carry out real-world experiments to verify our solution's performance. We use a desktop PC with an NVIDIA 1080ti graphic card as the edge infrastructure. For the cloud-based deep learning services, we choose Baidu Vision, Face++ object detection services and Amazon Rekognition. In the experiments, we use two datasets mentioned before in Subsection 3.6. The ImageNet dataset indicates daytime scenery, and the FLIR Thermal Dataset indicates nighttime scenery. Some important hyperparameters in our experiments are given in Table 1.

4.2 Dataset

We use two datasets, the ImageNet dataset and the FLIR Thermal Dataset. The ImageNet dataset is a Large-Scale Hierarchical Image, in which each node of the hierarchy is depicted by hundreds and thousands of images. Its images are mostly taken in the daytime. Therefore we use it as a daytime scenery. Usually, a surveillance camera captures colored pictures in the daytime and gray-scaled thermal images in the night. Therefore we choose a thermal image dataset to act as a nighttime image dataset. The FLIR Thermal Dataset is such a dataset having more than 14000 images collected by thermal sensors.

We use the ImageNet dataset in size reduction and accuracy performance experiment, DeepN-JPEG comparative experiment and end-to-end latency simulation experiment. Moreover, we use ImageNet and the FLIR Thermal Dataset alternately to simulate the scenery change in the *inference-estimation-querying-retraining* mechanism experiment.

4.3 Metrics

The default compression quality level for JPEG is usually 75 [35, 42]. Therefore we regard this as a reference value $c_{\text{ref}} = 75$ of the conventional benchmark.

In our experiments, we measure the compressed and reference image's file size to obtain the compression rate Δs. Since we do not have the real ground truth label of an image, we use the output $\vec{y}_{\text{ref}}$ from a reference image as the ground truth label, and calculate the relative top-5 accuracy $\mathcal{A}$ as the accuracy metric. The formula of $\mathcal{A}$ is presented in Subsection 3.1.

4.4 Upload Image Size Overhead

Figure 8 presents the upload traffic load of the training and inference phase. To be more intuitionistic, we plot the size overhead $\frac{\hat{s}}{\hat{s}_{\text{ref}}}$ as the y-axis where $\hat{s}$ is the real upload size of AdaCompress and $\hat{s}_{\text{ref}}$ is the benchmark upload size. Therefore $y \geq 1$ means that our solution uploads more data then benchmark, and $y < 1$ means the compression rate of AdaCompress. From Figure 8, we can see that as the training procedure runs, the upload image size decreases because the RL agent learns

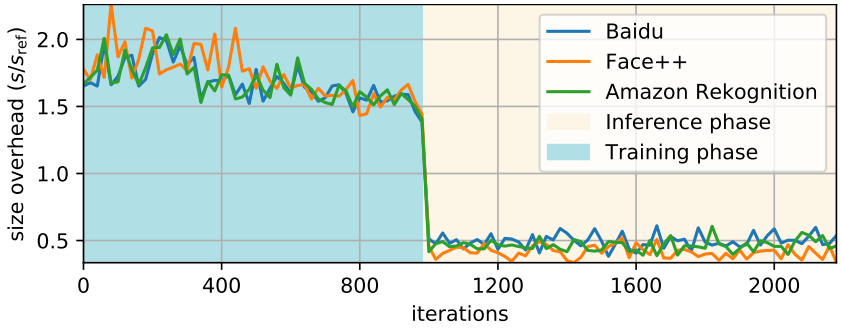


Figure 8. Size overhead in the training and inference phase

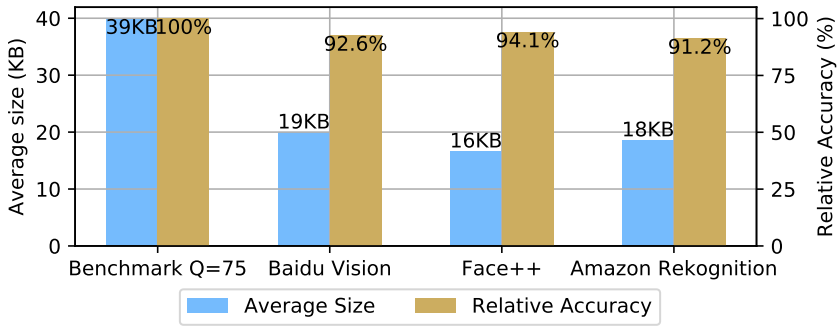


Figure 9. Average size and relative accuracy on different cloud services

to choose better compression quality levels to upload less data. In the training phase, to train the agent while remaining a convincing recognition result, AdaCompress has to upload the original image along with the compressed image to the cloud-end, obtaining the real result and the reward feedback. Therefore the upload traffic load is even higher than the conventional solution. But once the training phase has finished, the upload traffic load is lower than the benchmark. As shown in Figure 9, in the inference phase, AdaCompress's upload size is only 1/2 of the benchmark's.

4.5 Size Reduction and Accuracy Performance

Figure 9 presents the compression performance in the inference phase for each cloud service. We test AdaCompress on Face++, Baidu Vision and Amazon Rekognition. Comparing to the benchmark compression quality level, our solution can reduce the upload size by more than 1/2 for all tested cloud services, meanwhile maintain the relative accuracy that only decreases about 7% on average, proving the efficiency of our design.

4.6 DeepN-JPEG Comparative Experiment

Figure 10 presents a comparison performance between DeepN-JPEG and AdaCompress for each cloud service. As we can see, both DeepN-JPEG and AdaCompress cut down the upload size overhead more than 1/2. However, for all tested cloud services, AdaCompress's accuracy is higher than DeepN-JPEG's slightly. Compared with DeepN-JPEG's average accuracy is about 85% on three cloud services, AdaCompress achieves a better average accuracy of 93%. In a word, comparing

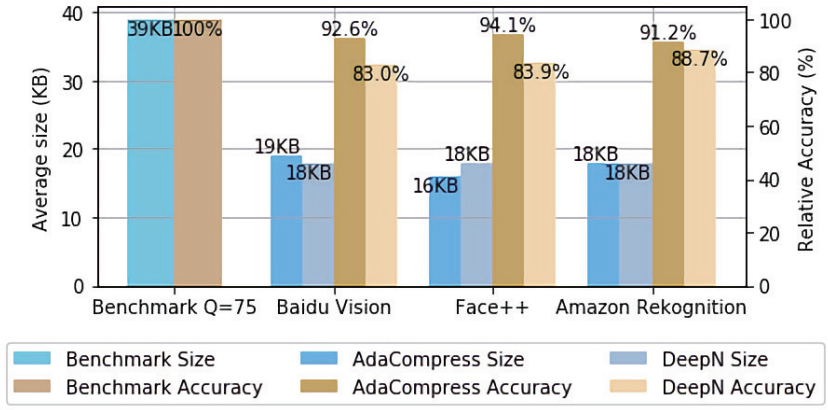


Figure 10. Comparative compression performance between DeepN-JPEG and AdaCompress

(1a) Origin Image (Q=75)
Baidu prediction = ["parrot"]

(1b) AdaCompress (choose Q=15)
Baidu prediction = ["parrot"]

(1c) DeepN-JPEG
Baidu prediction = ["parrot"]

(2a) Origin Image (Q=75)
Baidu prediction = ["cake"]

(2b) AdaCompress (choose Q=65)
Baidu prediction = ["cake"]

(2c) DeepN-JPEG
Baidu prediction = ["fossil"]

Figure 11. Comparative compressed images of DeepN-JPEG and AdaCompress

to the DeepN-JPEG framework, AdaCompress presents a similar size reduction performance but achieving higher inference accuracy for online computer vision-based services.

AdaCompress compresses images in a more adaptive manner rather than DeepN-JPEG. (i.e., the explanation why AdaCompress achieves a better accuracy). As shown in Figure 11, for picture 1a, compared to DeepN-JPEG, AdaCompress compresses the image at a more aggressive compression quality level of 15, reducing upload size overhead. On the contrary, for picture 2a of Figure 11,

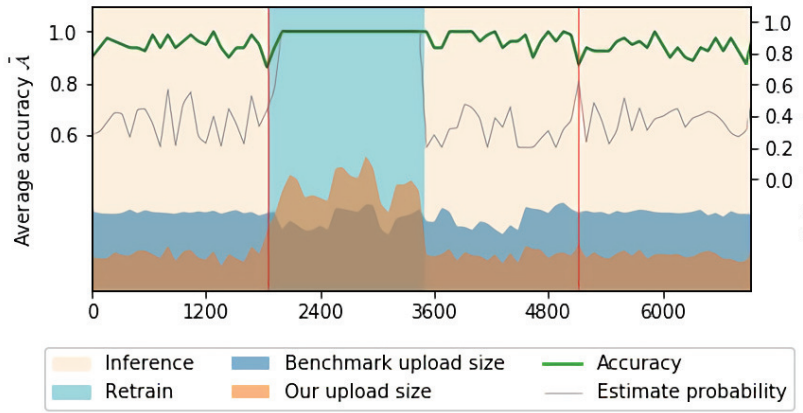


Figure 12. AdaCompress's reactions upon scenery change

DeepN-JPEG compresses the image with the same quantization table, but AdaComperss chooses a relatively higher compression quality level to preserve more details so that the backend deep learning model can still recognize the picture. Compared with DeepN-JPEG compresses all images with the same quantization table, AdaCompress chooses a low compression quality level for picture 1a and a relatively higher compression quality level for picture 2a based on the features of the input image.

Comparing to DeepN-JPEG, AdaCompress has two advantages and one disadvantage as following:

- AdaCompress and DeepN-JPEG both decrease the upload size overhead more than 1/2, but AdaCompress maintains higher inference accuracy.
- For different "sceneries" or cloud services, DeepN-JPEG compresses images with the same quantization table, while AdaCompress makes a more proper compression strategy adaptively.
- DeepN-JPEG re-designs the quantization table *locally*, while AdaCompress needs to upload reference images and compressed images to the cloud-end in the training phase, leading some upload size overhead at the beginning.

4.7 Adaptively Cope With the Scenery Change

To evaluate the efficiency of the *inference-estimation-querying-retraining* mechanism, we feed AdaCompress with a combined dataset whose first 2000 images from FLIR Thermal images, the next 3000 images randomly sampled from ImageNet and the last 2000 images from FLIR Thermal images. We adapt AdaCompress's current RL agent to FLIR nighttime scenery by training it on the FLIR dataset, and run AdaCompress on the combined dataset, observing AdaCompress's behaviors upon the scenery changes at step 2000 and 5000.

We illustrate AdaCompress's behaviors in Figure 12. The x-axis indicates steps, and the reference line indicates the scenery change. We plot AdaCompress's overall accuracy and the estimation probability p_{est}. At the bottom of Figure 12, we also plot the scaled upload size of AdaCompress and benchmark solution to illustrate the upload size overhead.

From Figure 12, we can see that AdaCompress can adaptively update the estimation probability p_{est}. When the overall accuracy decreases, AdaCompress would increase the estimation probability, trying to capture the scenery change. When the overall accuracy is stable and high enough, the estimation probability p_{est} decreases to reduce transmission.

Table 2. Latency of edge device to cloud service

Dataset	Solution	Average upload size	Inference latency	Transmission latency	End-to-end latency
ImageNet	Benchmark	42.68 KB	0 ms	12.35 ms	12.35 ms
ImageNet	AdaCompress	18.64 KB	2.09 ms	5.34 ms	7.43 ms
FLIR	Benchmark	44.66 KB	0 ms	12.93 ms	12.93 ms
FLIR	AdaCompress	17.07 KB	2.03 ms	4.94 ms	6.97 ms

Upon the first image scenery changes (i.e., night to day) shown as the first reference line in Figure 12, comparing to the earlier steps, the accuracy decreases dramatically and the estimation probability p_{est} raises to determine whether the scenery changes. The accuracy keeps dropping in the following steps, indicating that the current RL agent is no more suitable for the current input scenery. At the moment, AdaCompress should switch to the model querying state and re-load a new RL agent model from the model caching library. However, there is no model except the current RL agent at that time. Therefore, AdaCompress starts to re-train at once to adapt the RL agent to the current scenery. In the re-training phase, AdaCompress uses the reference image's prediction label $\vec{y}_{\text{ref}}$ as the output result. Therefore the accuracy $\mathcal{A}$ and p_{est} are locked to 1. After finishing re-training the agent in the daytime scenery, the trained agent is cached in the model caching library. In the following steps, sometimes the accuracy decreases accidentally, and the estimation probability p_{est} also raises. The accuracy is not lower than the accuracy threshold $\mathcal{A}_0$. Therefore the re-training phase would not be triggered again until the second image scenery changes.

Upon the second image scenery changes (i.e., day to night) shown as the second reference line in Figure 12, like the first scenery change, the accuracy decreases and p_{est} raises, indicating that the current RL agent is no more suitable again. AdaCompress switches to the model querying state at once and re-loads a new RL agent model (the initial model trained on the FLIR Thermal images) from the model caching library. When using the new agent in this scenery, the accuracy stops decreasing and maintains more than the accuracy threshold, indicating the current RL agent is suitable for the current scenery.

From Figure 12, we can also observe the upload size overhead in different phases. In the re-training phase, AdaCompress uploads more data than the conventional benchmark. But in the inference phase, AdaCompress's upload size is only half of the benchmark's. Especially once the second image scenery changes, AdaCompress achieves a low upload traffic load by re-loading a suitable RL agent model rather than re-training from scratch.

4.8 End-to-End Latency Simulation

Comparing to the conventional solution that uploads the image directly, in our solution, the image is passed to the RL agent firstly to estimate the compression quality level. Running this RL agent brings extra latency to the whole system. In this subsection, we evaluate the end-to-end latency.

We compute compressed file size for batches of images, and test the RL agent's inference time and the latency of uploading such compressed images. We respectively test the average inference latency on 1000 ImageNet images and 1000 FLIR images, and simulate the network bandwidth as 27.64 Mbps according to the global average fixed broadband upload speed [51] in Feb. 2019, to verify the end-to-end latency performance. The latency comparison is listed in Table 2.

Our solution brings in inference latency to the end-to-end latency, but the transmission latency is low by reducing the upload file size. In today's network architecture where the edge infrastructure's computational power is increasing significantly [27, 47], we can use the computing power of the edge infrastructure in exchange for the reduction of upload traffic load and transmission latency.

Compared with the benchmark solution, our solution effectively reduces the end-to-end latency on both ImageNet and FLIR dataset.

4.9 Overall Memory and Latency Overhead

4.9.1 Memory Overhead: The memory overhead is divided into two parts: the static occupied memory space and dynamic memory usage. Firstly, the static occupied memory space is the memory of the model's size. We use Keras to construct and save MobileNetV2 and the RL agent. We check file size of the saved model as the model's size. A RL agent model's size is 1 MB, and the feature extractor MobileNetV2's size is 8.9 MB. The model caching library stores the most recent five RL agents. The overall memory of the model's size is at most 13.9 MB. Secondly, the dynamic memory usage is the memory usage during AdaCompress runs. The overall memory usage mainly includes the memory buffer of transitions and the memory usage caused by the RL agent and feature extractor. When caching transitions of 1000 images in the memory buffer, it occupies about 500 KB. We use Psutil to measure the peak memory usage during AdaCompress runs. The peak memory usage is about 367 MB. Thirdly, in real-world edge-application scenarios, common edge infrastructures have enough memory to deploy this algorithm, such as Raspberry Pi 4 Model B's [39] memory is 2/4/8 GB, and Huawei Atlas 500 edge station's memory is 4/8 GB.

4.9.2 Training and Re-loading Latency: Although AdaCompress can be highly efficient to run on today's edge infrastructures, it would cause extra latency overhead, such as training and re-loading latency. In this subsection, we evaluate the latency overhead.

In our experiments, the convergence condition of training is that the recent average accuracy over 0.8 meanwhile the recent average reward over 0.45. The experiment results show the number of training images is 898 and 910 on ImageNet and FLIR, respectively. The time cost mainly depends on the frequency of sending images and the inference speed of the cloud service. If the end-user sends images continuously, and cloud service's inference speed is about 10 images per second, the time costs are 197.3s and 200.8s on ImageNet and FLIR, respectively. In the training phase, we need to upload reference images and compressed images. We consider the time cost of uploading reference images and obtaining the feedbacks as the benchmark latency. If we minus this benchmark latency, the real training time cost is not over 100s. It is worth to spend such upload size overhead and time cost at the beginning when we need to infer thousands of images.

The *inference-estimation-querying-retraining* mechanism would either re-load or re-train a RL agent intelligently when capturing the scenery change. The re-training time cost is similar to the training time cost which is described above. The latency of re-loading a RL agent model is about 1.85s. To *reduce* this latency, we use another process to re-load the model in our experiments. The mechanism still uses the old RL agent to choose a compression quality level before the re-loading process has finished.

5 CONCLUSION

To reduce the upload traffic load of deep learning applications, most researchers focus on modifying the deep learning model, but this does not apply to the industry because the backend deep learning model is usually inaccessible to users. We present a heuristic solution using a reinforcement learning agent to decide the proper compression quality level for each image according to the image's features and the backend service. Our experiment results show that for different backend deep learning cloud services and different input image "sceneries", using different compression strategies can significantly reduce the upload file size overhead while keeping comparable accuracy. Moreover, we design the *inference-estimation-querying-retraining* mechanism to cope with the input image scenery change and make a proper compression strategy for the current scenery. Our

experiment results show that once the scenery changes, the mechanism would either re-load a pre-trained agent or re-train a new agent intelligently to achieve low upload image size overhead while maintaining the inference accuracy.

REFERENCES

[1] Harsh Agrawal, Clint Solomon Mathialagan, Yash Goyal, Neelima Chavali, Prakriti Banik, Akrit Mohapatra, Ahmed Osman, and Dhruv Batra. 2015. Cloudcv: Large-scale distributed computer vision as a cloud service. In *Mobile cloud visual media computing*. Springer, 265–290.

[2] Amazon. 2019. Amazon Rekognition. https://aws.amazon.com/rekognition/.

[3] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. 2015. Fixed point optimization of deep convolutional neural networks for object recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 1131–1135.

[4] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. 2017. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 13, 3 (2017), 32.

[5] Baidu. 2019. Baidu AI Open Platform. https://ai.baidu.com/.

[6] Shumeet Baluja, David Marwood, and Nicholas Johnston. 2019. Task-specific color spaces and compression for machine-based object recognition. *Technical Disclosure Commons, (March 21, 2019)* (2019).

[7] M CALORE. 2010. Meet WebP, Google's New Image Format. Wired.

[8] Lahiru D Chamain, Sen-ching Samson Cheung, and Zhi Ding. 2019. Quannet: Joint Image Compression and Classification Over Channels with Limited Bandwidth. In *2019 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 338–343.

[9] Jianshu Chao, Hu Chen, and Eckehard Steinbach. 2013. On the design of a novel JPEG quantization table for improved feature detection performance. In *2013 IEEE International Conference on Image Processing*. IEEE, 1675–1679.

[10] Jianshu Chao and Eckehard Steinbach. 2011. Preserving SIFT features in JPEG-encoded images. In *2011 18th IEEE International Conference on Image Processing*. IEEE, 301–304.

[11] Kresimir Delac, Mislav Grgic, and Sonja Grgic. 2005. Effects of JPEG and JPEG2000 compression on face recognition. In *International Conference on Pattern Recognition and Image Analysis*. Springer, 136–145.

[12] ISO DIS. 1991. 10918-1. Digital Compression and Coding of Continuous-tone Still Images (JPEG). *CCITT Recommendation T* 81 (1991), 6.

[13] Samuel Dodge and Lina Karam. 2016. Understanding how image quality affects deep neural networks. In *2016 eighth international conference on quality of multimedia experience (QoMEX)*. IEEE, 1–6.

[14] Amir Erfan Eshratifar and Massoud Pedram. 2018. Energy and Performance Efficient Computation Offloading for Deep Neural Networks in a Mobile Cloud Computing Environment. In *Proceedings of the 2018 on Great Lakes Symposium on VLSI*. ACM, 111–116.

[15] Ivan Evtimov, Kevin Eykholt, Earlence Fernandes, Tadayoshi Kohno, Bo Li, Atul Prakash, Amir Rahmati, and Dawn Song. 2018. Robust physical-world attacks on deep learning models. In *Computer Vision and Pattern Recognition*.

[16] Face++. 2019. Face++ Cognitive Services. https://www.faceplusplus.com/.

[17] FLIR. 2018. FLIR Thermal Dataset. https://www.flir.com/oem/adas/adas-dataset-form/.

[18] Weifeng Ge and Yizhou Yu. 2017. Borrowing treasures from the wealthy: Deep transfer learning through selective joint fine-tuning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1086–1095.

[19] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. 2014. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115* (2014).

[20] Google.Inc. 2019. Google Edge TPU. https://cloud.google.com/edge-tpu/.

[21] Lionel Gueguen, Alex Sergeev, Ben Kadlec, Rosanne Liu, and Jason Yosinski. 2018. Faster neural networks straight from JPEG. In *Advances in Neural Information Processing Systems*. 3933–3944.

[22] Song Han, Huizi Mao, and William J Dally. 2015. A deep neural network compression pipeline: Pruning, quantization, huffman encoding. *arXiv preprint arXiv:1510.00149* 10 (2015).

[23] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*. 1135–1143.

[24] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. 2016. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 123–136.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[26] Pengfei Hu, Huansheng Ning, Tie Qiu, Yanfei Zhang, and Xiong Luo. 2017. Fog Computing-Based Face Identification and Resolution Scheme in Internet of Things. *IEEE Transactions on Industrial Informatics* 13, 4 (2017), 1910 – 1920.

[27] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. 2015. Mobile edge computing - A key technology towards 5G. *ETSI white paper* 11, 11 (2015), 1–16.
[28] Huawei. 2019. Huawei Atlas 500 Edge Station. https://e.huawei.com/en/products/cloud-computing-dc/servers/g-series/atlas-500.
[29] Loc N Huynh, Youngki Lee, and Rajesh Krishna Balan. 2017. Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 82–95.
[30] Kyuyeon Hwang and Wonyong Sung. 2014. Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1. In *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*. IEEE, 1–6.
[31] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J.Mars, and L. Tang. 2017. Neurosurgeon: collaborative intelligence between the cloud and mobile edge. In *ASPLOS*. ACM, 615–629.
[32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.
[33] Hongshan Li, Yu Guo, Zhi Wang, Shutao Xia, and Wenwu Zhu. 2019. AdaCompress: Adaptive Compression for Online Computer Vision Services. In *Proceedings of the 27th ACM International Conference on Multimedia*. 2440–2448.
[34] Hongshan Li, Chenghao Hu, Jingyan Jiang, Zhi Wang, Yonggang Wen, and Wenwu Zhu. 2018. JALAD: Joint Accuracy-And Latency-Aware Deep Structure Decoupling for Edge-Cloud Execution. In *24th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2018, Singapore, December 11-13, 2018*. 671–678. https://doi.org/10.1109/PADSW.2018.8645013
[35] Python Imaging Library. 2019. Image file formats. https://pillow.readthedocs.io/en/3.1.x/handbook/image-file-formats.html.
[36] Zihao Liu, Tao Liu, Wujie Wen, Lei Jiang, Jie Xu, Yanzhi Wang, and Gang Quan. 2018. DeepN-JPEG: a deep neural network favorable JPEG-based image compression framework. In *Proceedings of the 55th Annual Design Automation Conference*. ACM, 18.
[37] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).
[38] Jia D et al. Olga R. 2012. ImageNet Large Scale Visual Recognition Challenge 2012. http://image-net.org/challenges/LSVRC/2012/.
[39] Raspberry Pi. 2019. Raspberry Pi 4 Model B. https://www.raspberrypi.org/products/raspberry-pi-4-model-b/.
[40] Majid Rabbani. 2002. JPEG2000: Image compression fundamentals, standards and practice. *Journal of Electronic Imaging* 11, 2 (2002), 286.
[41] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/languageunsupervised/language understanding paper. pdf* (2018).
[42] rflynn. 2019. Lossy image optimization. https://github.com/rflynn/imgmin.
[43] Oren Rippel and Lubomir Bourdev. 2017. Real-time adaptive image compression. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2922–2930.
[44] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y
[45] Alireza Sadeghi, Gang Wang, and Georgios B Giannakis. 2019. Deep reinforcement learning for adaptive caching in hierarchical content delivery networks. *IEEE Transactions on Cognitive Communications and Networking* 5, 4 (2019), 1024–1033.
[46] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4510–4520.
[47] Mahadev Satyanarayanan. 2017. The emergence of edge computing. *Computer* 50, 1 (2017), 30–39.
[48] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. 2017. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision*. 618–626.
[49] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[50] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[51] SpeedTest. 2019. Speedtest Global Index. https://www.speedtest.net/global-index.

[52] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1–9.

[53] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. 2017. Lossy image compression with compressive autoencoders. *arXiv preprint arXiv:1703.00395* (2017).

[54] George Toderici, Sean M O'Malley, Sung Jin Hwang, Damien Vincent, David Minnen, Shumeet Baluja, Michele Covell, and Rahul Sukthankar. 2015. Variable rate image compression with recurrent neural networks. *arXiv preprint arXiv:1511.06085* (2015).

[55] George Toderici, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, Joel Shor, and Michele Covell. 2017. Full resolution image compression with recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5306–5314.

[56] Robert Torfason, Fabian Mentzer, Eirikur Agustsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. 2018. Towards image understanding from deep compression without decoding. *arXiv preprint arXiv:1803.06131* (2018).

[57] Gregory K Wallace. 1992. The JPEG still picture compression standard. *IEEE transactions on consumer electronics* 38, 1 (1992), xviii–xxxiv.

[58] Fangxin Wang, Feng Wang, Jiangchuan Liu, Ryan Shea, and Lifeng Sun. 2020. Intelligent Video Caching at Network Edge: A Multi-Agent Deep Reinforcement Learning Approach. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2499–2508.

[59] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. 2019. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems* (2019).

[60] Chen Zhong, M Cenk Gursoy, and Senem Velipasalar. 2018. A deep reinforcement learning-based framework for content caching. In *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 1–6.