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

Zhaoliang He, Hongshan Li, Zhi Wang, Shutao Xia, Wenwu Zhu

***Abstract*—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.**

***Index Terms*—Edge Computing, Reinforcement Learning, Adaptive Compression, Machine Learning Service**

I. 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 huge computational resource consumption for deep learning applications (e.g., inferring an image on VGG19 [2] requires 20 GFLOPs of GPU resource), in today's online computer vision-based applications, users 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, it is straightforward that an image should be compressed before one uploads 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. [3] 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 same 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 and input images? 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 [3]–[5], 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 structure 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 totally 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 [6] and commonly seen in adversarial neural network researches [7], [8].
- *Lack of well-labeled training data.* In our problem, one is not provided the well-labeled data on which image should

Z. He and W. Zhu are with Department of Computer Science and Techonology, Tsinghua University.

H. Li is with Tsinghua-Berkeley Shenzhen Institute, Tsinghua University.

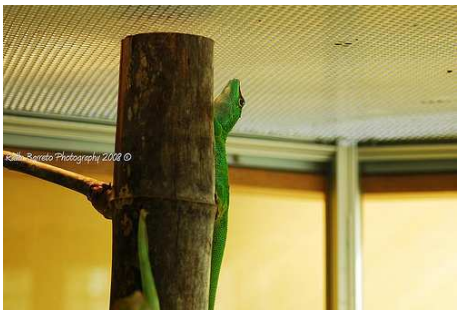
Z. Wang and S. Xia are with Tsinghua Shenzhen International Graduate School, Tsinghua University.

The preliminary version of this paper was published in ACM Multimedia 2019 [1]

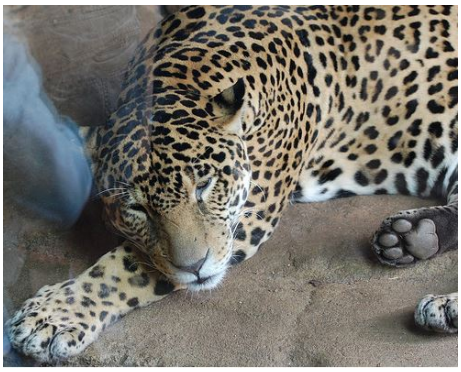
(1a) Q=75
Face++ prediction = [”donut”]

(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”]

Fig. 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 model still output correct label from a severely compressed image though they look very different

be compressed to 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 the proper compression quality level for an image to a computer vision model on the cloud, in an online manner. This paper is an extension of our earlier conference paper [1], 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 [9] 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

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

infrastructures (e.g., Google edge TPU [10], Huawei Atlas 500 edge station [11]).

▷ 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 reward that considers both accuracy and data size. Based on [1], we further propose an *explore-exploit* mechanism to let the agent switch between "sceneries". In particular, after deploying the agent, an *inference-estimate-retrain* solution is designed to restart the training 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. By analyzing the Deep Q-learning Network-based agent's behaviors using Grad-Cam [12], 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 deep learning services, including Amazon Rekognition [13], Face++ [14] and Baidu Vision [15], and 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 [3], 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 Sec. II. In Sec. III we present our framework and detailed design. We present our solution's performance in Sec. IV and conclude the paper in Sec. V.

II. Related Works

As online computer vision-based services have become the norm for today's applications [16], [17], many studies have been devoted to improving the cloud-based model execution, including model compression and data compression.

A. 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* [18]–[23], 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* [24]–[27], 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.

B. Data Compression

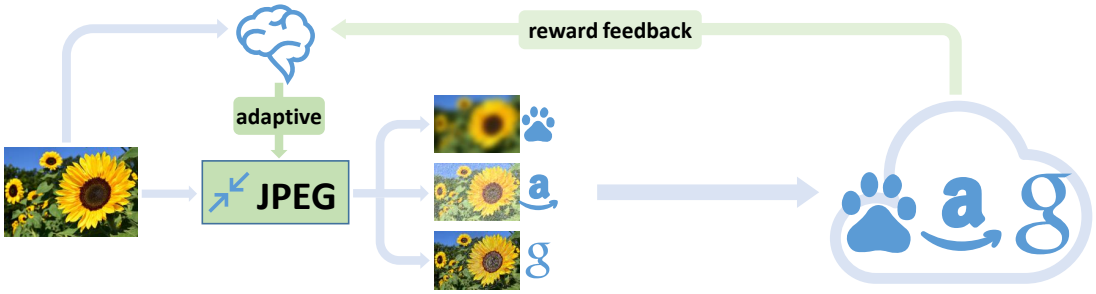
Data compression solutions study how to compress the original data (e.g., a video or image) to be inferred by the cloud-based deep learning model, so that less traffic is used to upload the data to improve inference speed. Conventional data compression solutions (e.g., JPEG [28], WebP [29] and JPEG2000 [30], etc.) and some recent neural network-based compression solutions [31]–[34] are initially designed for the Human-Visual System. In recent years, researchers found that the human visually optimized-based data compression solutions are not usually applicable to deep learning vision models. Delac et al. [6] observed that, in some cases, a higher compression quality level does not always deteriorates the model inference performance, even improves it slightly. Dodge et al. [35] further discovered that besides the JPEG compression, four types of quality distortions (blur, noise, contrast, and the JPEG2000 compression [30]) can also affect the inference performance of the deep learning models. Liu et al. [3] proposed 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 computer vision models' inference performance degradation at lower images' compression quality levels.

Based on these insights, Robert et al. [4] tried to train the deep neural network from the compressed representations of an auto-encoder. Chao et al. [36] proposed using variable quantization which is supported by the JPEG standard extension syntax [37] to compress the macroblocks in images. Furthermore, they [38] 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 with reduced image size. Liu et al. [3] 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. [39] proposed a joint optimization of image classification network coupled with the image quantization, achieving image size reduction of JPGE2000 [30] encoded images. Recently, Lionel et al. [5] presented a new type of neural network that infers directly from the discrete cosine transform (DCT) coefficients in the middle of the JPEG codec. Baluja et al. [40] 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, to *generate* the appropriate color space and/or compression schemes. To the best of our knowledge, we are the first to propose an adaptive compression configuration solution named AdaCompress [1] 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 extension paper, we improve previous *inference-estimate-retrain* mechanism to cut down the upload image size overhead effectively, add DeepN-JPEG comparative ex-

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



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

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

periment, and amend the manuscript significantly. Especially on the DeepN-JPEG comparative experiment, since Liu et al. [3] evaluated the DeepN-JPEG framework on ImageNet by using four state-of-the-art DNN models (AlexNet [41], VGG [42], GoogLeNet [43] and ResNet [44]) on local terminal 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 Sec. IV-C on three cloud-based deep learning services (Amazon Rekognition, Face++ and Baidu Vision), the same experiment configuration as AdaCompress's.

III. Detailed Design

A brief framework of AdaCompress is shown in Figure 2. Briefly, it is an reinforcement learning (Rl)-based system to train an agent to choose the 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, and *inference-estimation-querying-retraining* mechanism separately in the following subsections. We provide experimental details of all the hyperparameters in Sec. IV.

A. 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, and 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, and 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 [45].

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

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

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

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

Where $j = 1, ..., 5$ indicates the prediction labels at top-5 score, meaning that if anyone 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 s_{ic}. Especially for a reference compression quality level c_{ref}, the compressed file size is s_{ref}. Besides, the input image from a specific location usually belongs to a particular contextual group. For example, in daytime scenery, the input images are expected to have a bright background and high contrast while nighttime images are usually gray-scaled thermal images. Therefore, the agent in one place and one scenery does not need to know all the contextual features in all places and all "sceneries". We formulate this as contextual group $\mathcal{X}$. This contextual grouping concept is also discussed in the study [46].

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, and an image compressed at a reference level c_{ref} is also uploaded to the cloud-end to obtain $\vec{y}_{\text{ref}}$. Comparing the two uploaded instances $\{x_{\text{ref}}, x_c\}$ and cloud-end recognition results $\{\vec{y}_{\text{ref}}, \vec{y}_c\}$, we can obtain the reference file size s_{ref} and compressed file size s_c, therefore compute the file compression ratio $\Delta s = \frac{s_c}{s_{\text{ref}}}$ and accuracy metric $\mathcal{A}_c$.

B. RL Agent Design

The RL agent is expected to give a proper compression quality level c that minimizing the file size s_c while keeping the accuracy $\mathcal{A}$. 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 naive the Deep Q-learning Network can't work well in this task because of the following challenges:

- The state space of reinforcement learning is too large, and 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 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 well-trained neural network to extract the structural information of an input image. This is a commonly used strategy in training a deep neural network [47], [48]. Therefore instead of training an 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 [49] 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 the feature extractor $\mathcal{E}$ remains fixed.

C. reinforcement learning-based Framework

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

Based on this insight, we formulate the feature extractor's output $\mathcal{E}(J(\mathcal{X}, c))$ 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 $Q(\phi(\mathcal{E}(f_t)), c; \theta)$, and the optimal compression quality level at time t is $c_t = \text{argmax}_c Q(\phi(\mathcal{E}(f_t)), c; \theta)$ 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 $L_i(\theta_i) = \mathbb{E}_{s,c\sim\rho(\cdot)}\Big[\big(y_i - Q(s, c; \theta_i)\big)^2\Big]$ that changes at each iteration i where $s = \mathcal{E}(f_t)$, and $y_i = \mathbb{E}_{s'\sim\{\mathcal{X},M\}}\big[r + \gamma \max_{c'} Q(s', c'; \theta_{i-1}) \mid s, c\big]$ is the target for each iteration. Especially, r is the reward feedback, and $\rho(s, c)$ is a probability distribution over sequences s and compression quality level c [9]. 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.

In such a formulation, it is similar to the Deep Q-learning Network problem but not the same. 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 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. 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, and we list the parameters in Sec. IV.

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 sequence $s_1 = \mathcal{E}\big(J(x_1, c_1)\big), x_1 \in \mathcal{X}$ and $\phi_1 = \phi(f_1)$
for t = 1, K **do**
  With probability ϵ select a random compression level c_t otherwise select $c_t = \text{argmax}_c Q\Big(\phi\big(\mathcal{E}(f_t)\big), c; \theta\Big)$
  Compress image x_t at quality c_t and upload it to the cloud to get result $(\vec{y}_{\text{ref}}, \vec{y}_c)$ and calculate reward $r = R(\Delta s, \mathcal{A}_c)$
  Set $s_{t+1} = s_t$, generate c_t, x_{t+1} and preprocess $\phi_{t+1} = \phi\big(\mathcal{E}(f_{t+1})\big)$
  Store transition $(\phi_t, c_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
  if $t \mod T == 0$ and $t \geq T_{\text{start}}$ **then**
    Sample randomly mini-batch of transitions $(\phi_j, c_j, r_j, \phi_{j+1})$ from memory buffer $\mathcal{D}$
    Set $y_i = r_j + \gamma \max_{c'} Q(\phi_{j+1}, c'; \theta)$
    Compute decay exploration rate $\epsilon = \begin{cases} \mu_{\text{dec}} \cdot \epsilon & \text{if } \mu_{\text{dec}} \cdot \epsilon > \epsilon_{\min} \\ \epsilon_{\min} & \text{if } \mu_{\text{dec}} \cdot \epsilon \leq \epsilon_{\min} \end{cases}$
    Perform a gradient descent step on $\big(y_j - Q(\phi_j, c_j; \theta)\big)^2$ according to [9]
  end if
end for

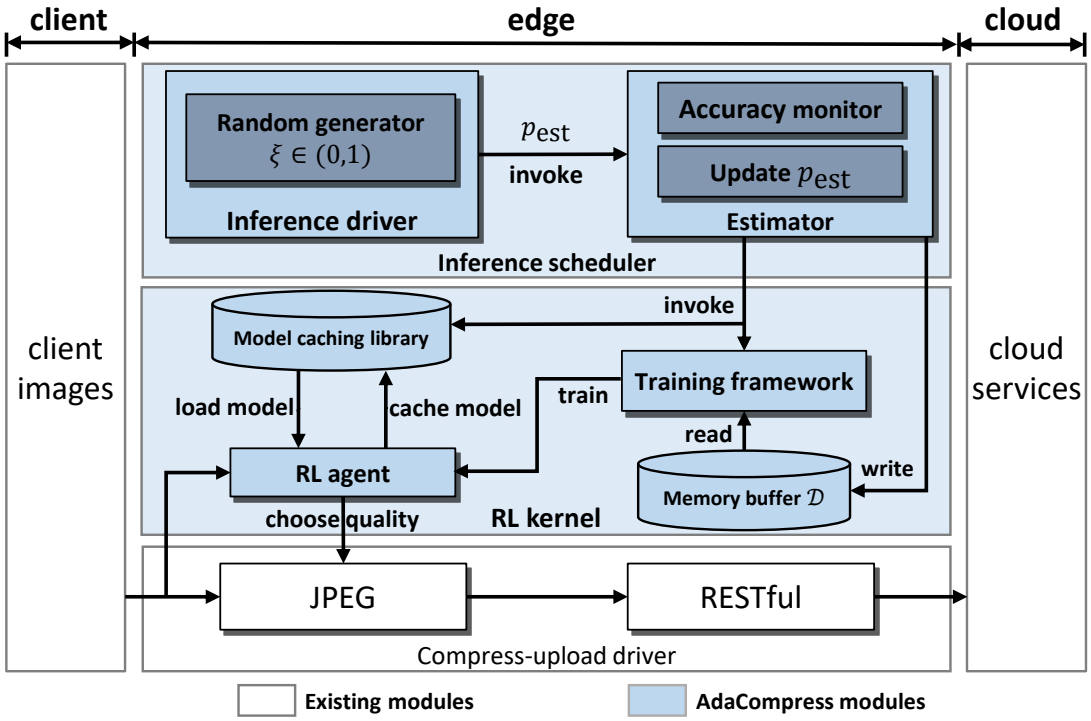


Fig. 3: Diagram of AdaCompress architecture

D. 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 defined compression rate $\Delta s = \frac{s_c}{s_{\text{ref}}}$ and accuracy metric $\mathcal{A}_c$ at compression quality level c. Basically, we want the agent to choose a proper compression quality level that minimizing the upload image size while remaining acceptable accuracy, therefore the overall reward r should be in proportion to the accuracy $\mathcal{A}$ while in inverse proportion to 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* $R(\Delta s, \mathcal{A})$.

E. 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 (RL) 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 according to the input image, then uploads the compressed image to cloud.

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 decreases. To cope with the scenery change, we invoke an estimator with a probability p_{est}. We do 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, and AdaCompress would upload the reference image x_{ref} along with the compressed image x_i to obtain $\vec{y}_{\text{ref}}$ and $\vec{y}_i$, therefore calculate $\mathcal{A}_i$, and save the transition $(\phi_i, c_i, r_i, \mathcal{A}_i)$ 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 an 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 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 back 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 query

a pre-trained model from the model caching library rather than re-training from scratch.

AdaCompress adaptively switch itself between four states. The switching policy is shown as Figure 4.

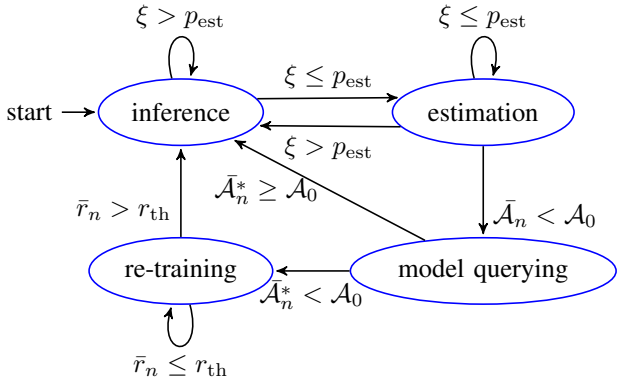


Fig. 4: State switching policy

*1) **Inference**:* For most times, AdaCompress runs in this state. In this state, only the compressed images are uploaded to the cloud to achieve minimum 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 the probability of p_{est}, meanwhile remain the inference state with the probability of $1 - p_{\text{est}}$.

*2) **Estimation**:* In this state, the reference image x_{ref} and compressed image x_i are uploaded to the cloud 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 $(\phi_i, c_i, r_i, \mathcal{A}_i)$ is logged in a memory buffer $\mathcal{D}$. When the average accuracy $\bar{\mathcal{A}}_n$ of the latest n steps is higher than the accuracy threshold $\mathcal{A}_0$, it stays in the estimation state with the probability of p_{est} or switches back into the inference state with the probability of $1 - p_{\text{est}}$. 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 loads 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 current agent's accuracy, so that to capture the scenery change on time; on the other hand, the estimator uploads the reference image x_{ref} along with the compressed image, therefore the upload image size overhead is greater than the conventional benchmark solution, causing higher 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 p_{est} solution. We first define that after running for N steps, the recent n steps' average accuracy is:

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

With this definition, an intuitive formulation of the change of p_{est} is in inverse proportion of the gradient of $\mathcal{A}$, meaning that when the recent accuracy decreases, the estimation probability p_{est} would increase. We formulate that $p'_{\text{est}} = p_{\text{est}} + \omega \nabla \bar{\mathcal{A}}$ where ω is a scaling factor. With this recursive formula, we have the general term of $p_{\text{est}} = p_0 + \omega \sum_{i=0}^{N} \nabla \bar{\mathcal{A}}_i$, with an initial estimation probability p_0.

*3) **Model Querying**:* The model querying state is designed to cope with the scenery change before re-training. It 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 back to the normal inference state using the loaded agent model. Otherwise, AdaCompress would switch into the re-training state and invokes 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, therefore cutting down the upload image size overhead. However, the agent caching strategy would cause a little storage overhead on local terminal devices.

*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.

F. Insight of RL Agent's Behavior

In the inference phase, the pre-trained RL agent chooses a proper compression quality level according to the input image's feature. The reference image is not uploaded to the cloud 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 of why the agent chooses a compression quality level).

*1) **Compression Level Choice Variation**:* In our experiment, we find that in different cloud application environments, the agent's chosen compression quality levels can be quite different. As shown in Figure 5, for Face++ and Amazon Rekognition, the agent's choices are concentrated at around $c = 15$, but for Baidu Vision, the agent's choices are distributed more evenly. Therefore, the "optimal" compression strategy should be 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 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.

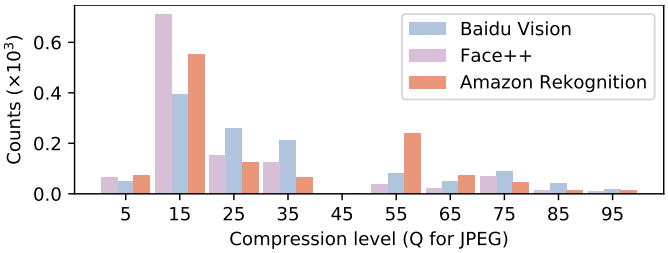


Fig. 5: Histogram of RL agent's best compression quality level selection for different cloud services

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 [50] whose images are mostly taken in the daytime, to act as a daytime scenery, and randomly select nighttime images from the FLIR Thermal Dataset [51] to form another dataset to act as a nighttime scenery. The histogram shown in Figure 6 points out that, for the ImageNet images, the agent prefers a lower compression quality level, but its choices are distributed more evenly. For the FLIR Thermal images, the agent's choices are more accumulated in some relatively high compression quality levels. We can see that, to maintain high accuracy, when the input image's contextual group $\mathcal{X}$ changes, the agent's compression quality level selection changes as well. This phenomenon presents that the agent can adaptively choose a proper compression quality level based on the input image's features.

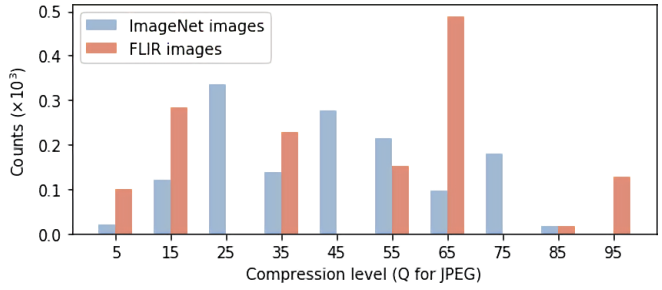


Fig. 6: Histogram of RL agent's best compression quality level selection for different input image "sceneries"

*2) **Attention Map Variation**:* To take 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 solution to present the importance map of a deep neural network, it is done 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 picks up compression quality level 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 highly, the agent selects lower compression quality levels based on the complex texture of the images. On the contrary, for pictures 2a – 2d, the agent chooses higher compression quality levels to preserve more details since its interest falls on some smooth regions. Especially for 1a and 2a, in picture 1a, the agent chooses a lower quality compression level based on the rough central region though there are smooth regions around it, but in picture 2a, the agent chooses a relatively higher compression quality level based on the surrounding smooth region rather than the central region.

IV. Evaluation

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

A. 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 deep learning services, we choose Baidu Vision, Face++ object detection services, and Amazon Rekognition. In the experiments, we use two datasets mentioned before in Sec. III-F, ImageNet dataset indicating daytime scenery and the FLIR Thermal Dataset indicating nighttime scenery. Some important hyperparameters in our experiments are given in Table I.

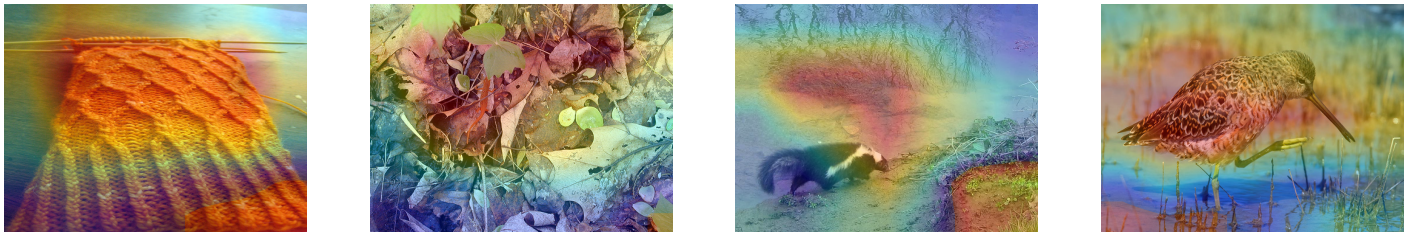
notation	value	notation	value
c_{ref}	75	K	1000
$\epsilon_{\min}$	0.02	p_0	0.2
γ	0.95	ω	-3
μ_{dec}	0.99	T	5
r_{th}	0.45	n	10

TABLE I: Experiment parameter settings

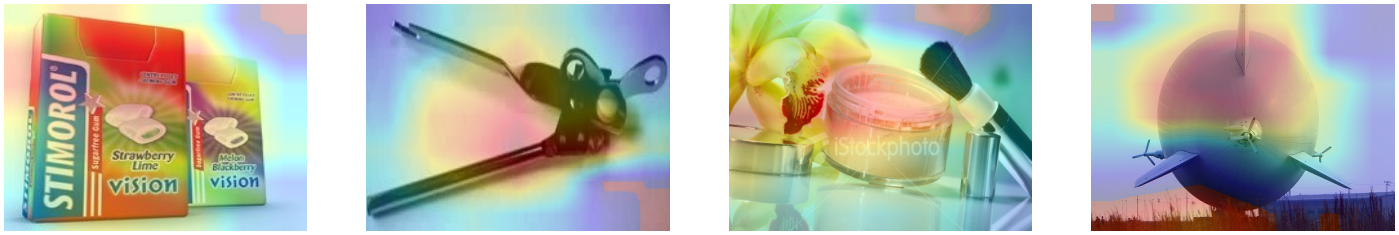
B. Dataset

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, and therefore we use it as a daytime scenery. Usually, a surveillance camera captures colored images in the daytime and gray-scaled thermal images in the nighttime. 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. Moreover, we use



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



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

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

ImageNet and the FLIR Thermal Dataset alternately to simulate the scenery change in the *inference-estimation-querying-retraining* mechanism experiment, and analyze the RL agent's best compression strategy selection in different input image "sceneries".

C. Metrics

The default compression quality level for JPEG is usually 75 [52], [53], therefore we regard this as a typical value $c_{\text{ref}} = 75$ of the conventional benchmark.

In our experiments, we measure the compressed and original image's file size to obtain the compression rate Δs. Since we don't 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 Sec. III-A.

D. Upload Image Size Overhead

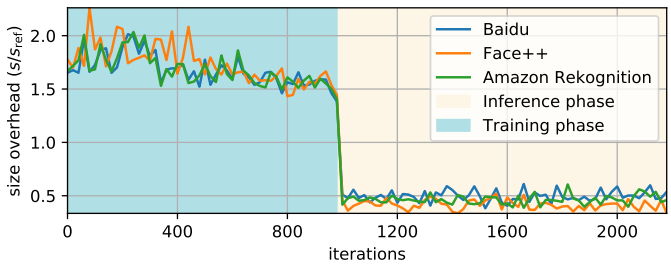


Fig. 8: Size overhead in training and inference phase

Figure 8 presents the upload traffic load of the training and inference phase, to be more intuitionistic, we plot the size overhead $\frac{s}{s_{\text{ref}}}$ as the y-axis where s is the real upload size of AdaCompress, s_{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 to choose better compression quality levels to upload fewer data. In the training phase, to train the agent while remaining a convincing recognition result, we have to upload the original image to the cloud to get the real result, along with the compressed image to obtain 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.

E. Size Reduction and Accuracy Performance

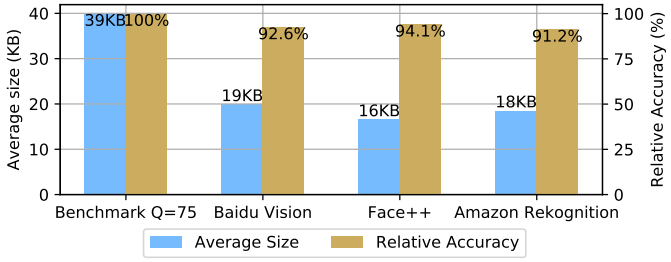


Fig. 9: Average size and relative accuracy on different cloud services

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 conventional compression level, for all tested cloud services, our solution can reduce the upload size by more than 1/2, meanwhile, the relative accuracy, indicated by brown bars, only decreases about 7% on average, proving the efficiency of our design.

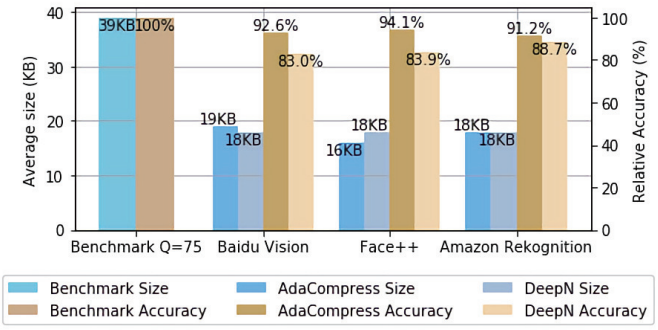


Fig. 10: Comparative compression performance between DeepN-JPEG and AdaCompress

F. 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, while AdaCompress achieves a better average accuracy of 93%. In a word, comparing to the DeepN-JPEG framework, AdaCompress presents a similar upload traffic load 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 of why AdaCompress achieves a better accuracy). As shown in Figure 11, for picture 1a, compared to DeepN-JPEG, AdaCompress compresses the image at the compression quality level of 15, a more aggressive compression quality level to reduce upload size overhead. On the contrary, for picture 2a of Figure 11, DeepN-JPEG compresses the image with the same quantization table, but AdaComperss chooses a higher compression quality level to preserve more details so that the backend deep learning model can still recognize the picture. Being different from the DeepN-JPEG framework compresses all images with the same quantization table, our RL agent chooses a lower compression quality level for picture 1a and a relatively higher compression quality level for picture 2a based on the feature of the input image.

Comparing to DeepN-JPEG, AdaCompress has three 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.
- DeepN-JPEG requires the original dataset information to re-design the quantization table, while AdaCompress chooses compression quality level adaptively without any pre-knowledge of the original dataset.
- When the scenery changes, DeepN-JPEG still compresses images with the same quantization table, while AdaCompress's *inference-estimation-querying-retraining* mechanism captures the scenery change and invokes either model querying method or retraining kernel to *generate* a more suitable compression selection strategy for the current scenery.
- DeepN-JPEG re-designs the quantization table *locally*, while AdaCompress needs to upload origin images and compressed images to the cloud in the training phase, leading some upload size overhead at the beginning.

G. 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 later 3000 images randomly sampled from ImageNet, and the last 2000 images from FLIR Thermal images. We adapt AdaCompress's current RL agent to FLIR Thermal nighttime scenery by training it on the FLIR dataset, and we run AdaCompress on the combined dataset, observing AdaCompress's behavior upon the scenery changes at step 2000 and 5000.

We illustrate AdaCompress's behavior in Figure 12, the x-axis indicates steps, the vertical red line means the dataset change (i.e. scenery change). We plot AdaCompress's overall accuracy as the green line and the estimation probability p_{est} as the gray line. At the bottom of Figure 12, we also plot the scaled upload size of AdaCompress and benchmark solution to illustrate the upload size overhead in the inference phase.

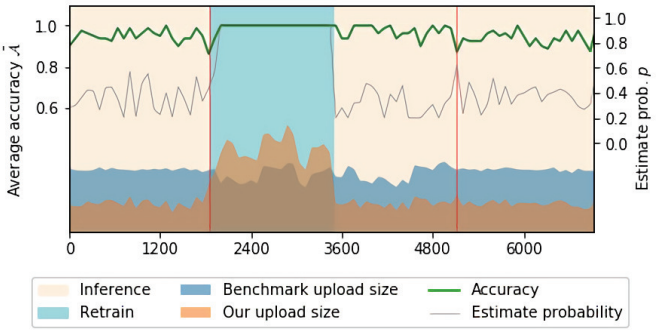


Fig. 12: AdaCompress's reaction upon scenery change

From Figure 12 we can see that AdaCompress can adaptively update the estimation probability p_{est}, usually, 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.

Upon the first data scenery changes (i.e., night to day) shown as the first vertical red line in Figure 12, comparing to the earlier steps, the accuracy decreases dramatically and therefore p_{est} raises to determine whether the scenery changes. The accuracy keeps dropping in the following estimations, indicating that the current RL agent is no more suitable for the current input scenery. At the moment, AdaCompress should switch into 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 in the model caching library at that time. Therefore, AdaCompress starts to re-train

(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"]

Fig. 11: Comparative compressed images of DeepN-JPEG and AdaCompress

at once, to adapt the RL agent into the current scenery. The re-training steps are shown as the light-blue region in Figure 12. 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 cachling library, and in the following iterations, sometimes the accuracy decreases accidentally, and the estimation probability p_{est} also raises to get more samples. 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 data scenery changes.

Upon the second data scenery changes (i.e., day to night) shown as the second vertical red 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 into the model querying state at once and re-loades a new RL agent model (i.e., 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 above the accuracy threshold, indicating the current RL agent is suitable for the current scenery.

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

H. 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 first to estimate the compression level. Running this RL agent brings extra latency to the whole system. In this subsection, we evaluate this latency overhead.

We test the RL agent's inference time and compressed file size for batches of images, and simulate the latency of uploading such compressed images. We test the average inference latency on 1000 ImageNet images and simulate the network bandwidth as 27.64 Mbps according to the global average fixed broadband upload speed [54] in Feb. 2019, to verify the end-to-end latency performance. The latency comparison is listed in Table II.

	Benchmark	AdaCompress
Average upload size	42.68 KB	18.46 KB
Inference latency	0 s	2.09 ms
Transmission latency	12.35 ms	5.34 ms
Overall latency	12.35 ms	7.43 ms

TABLE II: Latency between image upload and inference result feedback

Our solution brings in inference latency to the end-to-end latency, but the transmission latency is much lower by shrinking the upload file size. In today's network architecture where the edge infrastructure's computational power is increasing significantly [55], [56], we can use the computing power of the edge infrastructure in exchange for the reduction of upload traffic load and transmission latency.

V. 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 model is usually inaccessible for users. We present a heuristic solution using an reinforcement learning agent to decide the proper compression quality level for each image, according to the input image and the backend service. Our experiments show that for different backend deep learning cloud services and different input image "sceneries", using different quality selection strategy 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 selection strategy for the current scenery. Our experiment result shows that once the scenery changes, the mechanism would either re-load a pre-trained agent model or re-train a new agent intelligently to achieve lower upload image size overhead while maintaining the inference accuracy.

References

[1] H. Li, Y. Guo, Z. Wang, S. Xia, and W. Zhu, "Adacompress: Adaptive compression for online computer vision services," in *Proceedings of the 27th ACM International Conference on Multimedia*, 2019, pp. 2440–2448.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[3] Z. Liu, T. Liu, W. Wen, L. Jiang, J. Xu, Y. Wang, and G. Quan, "Deepn-jpeg: a deep neural network favorable jpeg-based image compression framework," in *Proceedings of the 55th Annual Design Automation Conference*. ACM, 2018, p. 18.

[4] R. Torfason, F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. Van Gool, "Towards image understanding from deep compression without decoding," *arXiv preprint arXiv:1803.06131*, 2018.

[5] L. Gueguen, A. Sergeev, B. Kadlec, R. Liu, and J. Yosinski, "Faster neural networks straight from jpeg," in *Advances in Neural Information Processing Systems*, 2018, pp. 3933–3944.

[6] K. Delac, M. Grgic, and S. Grgic, "Effects of jpeg and jpeg2000 compression on face recognition," in *International Conference on Pattern Recognition and Image Analysis*. Springer, 2005, pp. 136–145.

[7] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE transactions on neural networks and learning systems*, 2019.

[8] I. Evtimov, K. Eykholt, E. Fernandes, T. Kohno, B. Li, A. Prakash, A. Rahmati, and D. Song, "Robust physical-world attacks on deep learning models," in *Computer Vision and Pattern Recognition*, 2018.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[10] Google.Inc., "Google edge tpu," https://cloud.google.com/edge-tpu/, 2019.

[11] Huawei, "Huawei atlas 500 edge station," https://e.huawei.com/en/products/cloud-computing-dc/servers/g-series/atlas-500, 2019.

[12] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 618–626.

[13] Amazon, "Amazon rekognition," https://aws.amazon.com/rekognition/, 2019.

[14] Face++, "Face++ cognitive services," https://www.faceplusplus.com/, 2019.

[15] Baidu, "Baidu ai open platform," https://ai.baidu.com/, 2019.

[16] L. N. Huynh, Y. Lee, and R. K. Balan, "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, 2017, pp. 82–95.

[17] H. Agrawal, C. S. Mathialagan, Y. Goyal, N. Chavali, P. Banik, A. Mohapatra, A. Osman, and D. Batra, "Cloudcv: Large-scale distributed computer vision as a cloud service," in *Mobile cloud visual media computing*. Springer, 2015, pp. 265–290.

[18] S. Han, H. Mao, and W. J. Dally, "A deep neural network compression pipeline: Pruning, quantization, huffman encoding," *arXiv preprint arXiv:1510.00149*, vol. 10, 2015.

[19] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[20] S. Anwar, K. Hwang, and W. Sung, "Fixed point optimization of deep convolutional neural networks for object recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1131–1135.

[21] K. Hwang and W. Sung, "Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1," in *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*. IEEE, 2014, pp. 1–6.

[22] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," *arXiv preprint arXiv:1412.6115*, 2014.

[23] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, p. 32, 2017.

[24] A. E. Eshratifar 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*. ACM, 2018, pp. 111–116.

[25] H. Li, C. Hu, J. Jiang, Z. Wang, Y. Wen, and W. Zhu, "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*, 2018, pp. 671–678. [Online]. Available: https://doi.org/10.1109/PADSW.2018.8645013

[26] P. Hu, H. Ning, T. Qiu, Y. Zhang, and X. Luo, "Fog computing-based face identification and resolution scheme in internet of things," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 4, pp. 1910 – 1920, 2017.

[27] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J.Mars, and L. Tang, "Neurosurgeon: collaborative intelligence between the cloud and mobile edge," in *ASPLOS*. ACM, 2017, pp. 615–629.

[28] G. K. Wallace, "The jpeg still picture compression standard," *IEEE transactions on consumer electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992.

[29] M. CALORE, "Meet webp, google's new image format. wired," 2010.

[30] M. Rabbani, "Jpeg2000: Image compression fundamentals, standards and practice," *Journal of Electronic Imaging*, vol. 11, no. 2, p. 286, 2002.

[31] G. Toderici, D. Vincent, N. Johnston, S. Jin Hwang, D. Minnen, J. Shor, and M. Covell, "Full resolution image compression with recurrent neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5306–5314.

[32] L. Theis, W. Shi, A. Cunningham, and F. Huszár, "Lossy image compression with compressive autoencoders," *arXiv preprint arXiv:1703.00395*, 2017.

[33] G. Toderici, S. M. O'Malley, S. J. Hwang, D. Vincent, D. Minnen, S. Baluja, M. Covell, and R. Sukthankar, "Variable rate image compression with recurrent neural networks," *arXiv preprint arXiv:1511.06085*, 2015.

[34] O. Rippel and L. Bourdev, "Real-time adaptive image compression," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2922–2930.

[35] S. Dodge and L. Karam, "Understanding how image quality affects deep neural networks," in *2016 eighth international conference on quality of multimedia experience (QoMEX)*. IEEE, 2016, pp. 1–6.

[36] J. Chao and E. Steinbach, "Preserving sift features in jpeg-encoded images," in *2011 18th IEEE International Conference on Image Processing*. IEEE, 2011, pp. 301–304.

[37] I. DIS, "10918-1. digital compression and coding of continuous-tone still images (jpeg)," *CCITT Recommendation T*, vol. 81, p. 6, 1991.

[38] J. Chao, H. Chen, and E. Steinbach, "On the design of a novel jpeg quantization table for improved feature detection performance," in *2013 IEEE International Conference on Image Processing*. IEEE, 2013, pp. 1675–1679.

[39] L. D. Chamain, S.-c. S. Cheung, and Z. Ding, "Quannet: Joint image compression and classification over channels with limited bandwidth," in *2019 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2019, pp. 338–343.

[40] S. Baluja, D. Marwood, and N. Johnston, "Task-specific color spaces and compression for machine-based object recognition," *Technical Disclosure Commons, (March 21, 2019)*, 2019.

[41] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[42] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[43] C. Szegedy, 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*, 2015, pp. 1–9.

[44] K. He, 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*, 2016, pp. 770–778.

[45] J. D. e. a. Olga R, "Imagenet large scale visual recognition challenge 2012," http://image-net.org/challenges/LSVRC/2012/, 2012.

[46] S. Han, 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*. ACM, 2016, pp. 123–136.

[47] W. Ge and Y. Yu, "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*, 2017, pp. 1086–1095.

[48] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "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.

[49] M. Sandler, 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*, 2018, pp. 4510–4520.

[50] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[51] FLIR, "Flir thermal dataset," https://www.flir.com/oem/adas/adas-dataset-form/, 2018.

[52] P. I. Library, "Image file formats," https://pillow.readthedocs.io/en/3.1.x/handbook/image-file-formats.html, 2019.

[53] rflynn, "Lossy image optimization," https://github.com/rflynn/imgmin, 2019.

[54] SpeedTest, "Speedtest global index," https://www.speedtest.net/global-index, 2019.

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

[56] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing - a key technology towards 5g," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.