AdaCompress: Adaptive Compression for Online Computer Vision Services

Hongshan Li
Tsinghua-Berkeley Shenzhen Institute, Tsinghua University
lhs17@mails.tsinghua.edu.cn

Yu Guo
Graduate School at Shenzhen, Tsinghua University
guoy18@mails.tsinghua.edu.cn

Zhi Wang*
Graduate School at Shenzhen, Tsinghua University
Peng Cheng Laboratory
wangzhi@sz.tsinghua.edu.cn

Shutao Xia
Graduate School at Shenzhen, Tsinghua University
xiast@sz.tsinghua.edu.cn

Wenwu Zhu*
Tsinghua-Berkeley Shenzhen institute, Department of Computer Science and Technology, Tsinghua University
wwzhu@tsinghua.edu.cn

ABSTRACT

With the growth of computer vision based applications and services, an explosive amount of images have been uploaded to cloud servers which host such computer vision algorithms, usually in the form of deep learning models. JPEG has been used as the *de facto* compression and encapsulation method before one uploads the images, due to its wide adaptation. 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 including InceptionNet, ResNet, etc. Knowing this, designing a better JPEG configuration for online computer vision services is still extremely challenging: 1) Cloud-based computer vision models are usually a black box to end-users; thus it is difficult to design JPEG configuration without knowing their model structures. 2) JPEG configuration has to change when different users use it. In this paper, we propose a reinforcement learning based JPEG configuration framework. In particular, we design an agent that adaptively chooses the compression level according to the input image's features and backend deep learning models. Then we train the agent in a reinforcement learning way to adapt it for different deep learning cloud services that act as the *interactive training environment* and feeding a reward with comprehensive consideration of accuracy and data size. In our real-world evaluation on Amazon Rekognition, Face++ and Baidu Vision, our approach can reduce the size of images by 1/2 – 1/3 while the overall classification accuracy only decreases slightly.

*corresponding authors

MM '19, October 21–25, 2019, Nice, France

ACM ISBN 978-1-4503-6889-6/19/10…$15.00
https://doi.org/10.1145/3343031.3350874

CCS CONCEPTS

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

KEYWORDS

edge computing;reinforcement learning;data compression;online computer vision services

ACM Reference Format:
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 (MM '19), October 21–25, 2019, Nice, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3343031.3350874

1 INTRODUCTION

With the great success of deep learning in computer vision, this decade has witnessed an explosion of deep learning based computer vision applications. Because of the huge computational resource consumption for deep learning applications (e.g., inferring an image on VGG19 [37] requires 20 GFLOPS GPU resource), in today's computer vision 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 uploading traffic burden. For example, a picture taken by a cellphone at the resolution of 3968×2976 when saved as JPEG format at the default compression level, has a size up to 3MB.

To reduce the upload traffic, 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, because JPEG was originally designed for human vision system. Liu et al. [27] showed that by modifying the *quality level* in the default JPEG configuration, by retraining it on the original dataset, one can compress an image to a smaller version while maintaining the inference accuracy for a fixed deep computer vision algorithm. We then raise an intuitive question: to make it practically useful, can we improve the JPEG configuration adaptively for different cloud computer vision services, without any pre-knowledge of the original model and dataset?

Our answer to this question is a new learning-based compression methodology for today's cloud computer vision services. We tackle the following challenges in our design.

- **Lack of information about the cloud computer vision models.** Different from the studies [15, 27, 42], in which the computer vision models are available so that one can adjust the JPEG configuration according to the model structure or retrain the parameters in it, e.g., one can greedily search a gradient descent to reach an optimal compression level in JPEG. In our study, however, the details of the online cloud computer vision model are inaccessible.
- **Different cloud computer vision models need different JPEG configurations.** As an adaptive JPEG configuration solution, we target to provide a solution that is adaptive to different cloud computer vision services, i.e., it can *generate* JPEG configuration for different models. However, today's cloud computer vision algorithms, based one deep and convolutional computations, are quite hard to understand. The same compression level could lead to 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 models give different inference results, only because they are compressed at different quality levels. And such 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 [7] and commonly seen in adversarial neural network researches [10, 43].
- **Lack of well-labeled training data.** In our problem, one is not provided the well-labeled data on which image should 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 deep reinforcement learning (DRL) based solution, AdaCompress, to choose the proper compression level for an image to a computer vision model on the cloud, in an online manner. We open-sourced[1] our JPEG configuration module that works with today's cloud computer vision APIs upon acceptance of this paper. In particular, our contributions are summarized as follows:

- First, we design an interactive training environment that can be applied to different computer vision cloud services at different times, then we propose a deep Q neural network agent to evaluate and predict the performance of a compression level on an input image. In real-world application scenarios, this deep Q neural network can be highly efficient to run on today's edge infrastructures (e.g., Google edge TPU [14], Huawei Atlas 500 edge station [21]).
- Second, we build up a reinforcement learning framework to train the deep Q network in the above environment. By feeding the agent with carefully designed reward comprehensively considering accuracy and data size, the agent can learn to choose a proper compression level for an input image after iteratively interacting with the environment. To make the solution adaptive to the changing input images, we propose an explore-exploit mechanism to adapt the agent to different "scenery" online. After deploying the deep Q agent, an inference-estimate-retrain mechanism is designed to restart the training procedure once the scenery changes, and the existing running Q agent cannot guarantee stable accuracy performance.
- Finally, we provide analysis and insights on our design. We analyze the Q network's behavior by introducing Grad-Cam [36], and we explain why the Q network chooses a specific compression level, and provide some general patterns. Generally speaking, images that contain large smooth areas are more sensitive to compression, while the images with complex textures are more robust to compression when shown to deep learning models. We evaluate our system on some most popular cloud deep learning services, including Amazon Rekognition [2], Face++ [11] and Baidu Vision [5], and show that our design can reduce the uplink traffic load by up to 1/2 while maintaining comparable overall accuracy.

[1] https://github.com/hosea1008/AdaCompress

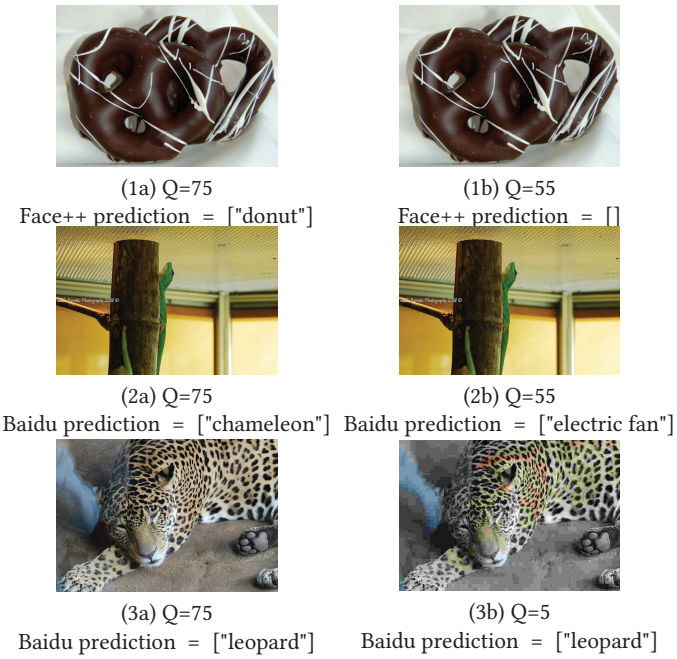
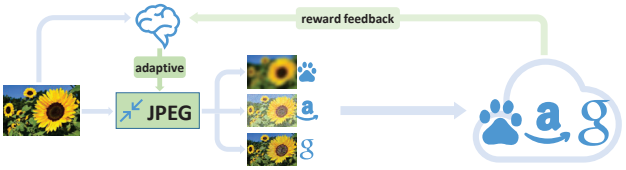


Figure 1: The prediction of a deep learning model is not completely related to the input image's quality, making it difficult to use a fixed compression quality for all images. For image 1a, 1b and 2a, 2b, minor changes cause different predictions though they are visually similar; for image 3a and 3b,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

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

(a) Conventional solution:
fixed user-defined compression 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

2 DETAILED DESIGN

A brief framework of AdaCompress is shown in Figure 2. Briefly, it is a DRL (deep reinforcement learning) based system to train an agent to choose the proper quality level c for one image to be compressed by JPEG. We will discuss the formulation, agent design, reinforcement learning framework, reward feedback, and retrain mechanism separately in the following subsections. We will provide experimental details of all the hyperparameters in Sec. 3.

2.1 Problem formulation

Without loss of generality, we denote the cloud 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 all reference input $x \in X_{\text{ref}}$. We use this $\vec{y}_{\text{ref}}$ as the ground truth labels, and for each image x_c compressed at quality c, we have $\vec{y}_c = M(x_c)$. Therefore, we have an accuracy metric $\mathcal{A}_c$ by comparing $\vec{y}_{\text{ref}}$ and $\vec{y}_c$. To be general, we use the top-5 accuracy as the following $\mathcal{A}$, the same as the classification metric of ILSVRC2012 [29].

$$\begin{aligned}\mathcal{A} &= \sum_k \min_j d(l_j, g_k) \\ & l_j \in \vec{y}_c, \quad j = 1, \ldots, 5 \\ & g_k \in \vec{y}_{\text{ref}}, \quad k = 1, \ldots, \text{length}(\vec{y}_{\text{ref}}) \\ & d(x, y) = 1 \text{ if } x = y \text{ else } 0\end{aligned}$$

Where $j = 1, \ldots, 5$ indicating the prediction labels at top-5 score, $k = 1, \ldots, \text{length}(\vec{y}_{\text{ref}})$ means that if anyone of the top-5 predicted labels matches one of the predictions from $\vec{y}_{\text{ref}}$, it is regarded as a correct prediction. To be general, we stipulate that for a cloud deep learning service, we cannot get the deep model's in-layer details (e.g., softmax probabilities) 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 c, it outputs a compressed file f_{ic} at the size of s_{ic}, for a reference compression level c_{ref}, the compressed file size is s_{ref}. Besides, images input from a specific location usually belong to a particular contextual group. For example, in an indoor scenery, the user input is less likely to have the images of the ocean, airplanes, and dolphins but more likely to have furniture and so on. Therefore, the agent at one place does not need to know all the contextual features in all places. We formulated this as contextual group $\mathcal{X}$. This contextual grouping concept is also discussed in [18].

Initially, the agent tries different compression level $c_{\min} < c < c_{\max}, c \in \mathbb{N}$ 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 to obtain $\vec{y}_{\text{ref}}$. Comparing the two uploaded instances $\{x, x_c\}$ and cloud recognition results $\{\vec{y}_{\text{ref}}, \vec{y}_c\}$, we can have the reference file size s_{ref} and compressed file size s_c and therefore the file compression ratio $\Delta s = \frac{s_c}{s_{\text{ref}}}$ and accuracy metric $\mathcal{A}_c$.

2.2 DRL agent design

The DRL agent is expected to give a proper compression level c that minimizing the file size s_c while keeping the accuracy $\mathcal{A}$. For the DRL agent, the input features are continuous numerical vectors, and the expected output are discrete quality levels c, therefore we can use DQN (Deep Q Network) [28] as the DRL agent. But naive DQN 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 DRL 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.
- DRL starts training from random trials, and starts learning after it found a better 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 [12, 30]. Therefore instead of training a DRL 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 [34] as the image feature extractor $\mathcal{E}(\cdot)$ for its efficiency in image classification and lightweight. The Q network ϕ is connected to the feature extractor's last convolution layer, therefore the output of $\mathcal{E}$ is the input of ϕ. We update the RL agent's policy by changing the parameters of Q network ϕ while the feature extractor $\mathcal{E}$ remains fixed.

2.3 Reinforcement learning framework

In a specific scenery where the user input 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 c as discrete *actions*. In our system, to accelerate training, we define 10 discrete actions to indicate 10 quality levels of JPEG ranging from 5, 15, ..., 95. We denote the *action-value function* as $Q(\phi(\mathcal{E}(f_t)), c; \theta)$, then the optimal compression level at time t is $c_t = \text{argmax}_c Q(\phi(\mathcal{E}(f_t)), c; \theta)$ where θ indicates the parameters of Q network ϕ. In such reinforcement learning formulation, the training phase is to minimize a 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 $s = \mathcal{E}(f_t)$, and $y_i = \mathbb{E}_{s'\sim\{\mathcal{X},M\}}\left[r + \gamma \max_{c'} Q(s', c'; \theta_{i-1}) \mid s, c\right]$ is the target for iteration i, r is the feedback reward and $\rho(s, c)$ is a probability distribution over sequences s and quality level c [28]. When minimizing the distance of the 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 formulation, it is similar to DQN problem but not the same. Different from conventional reinforcement learning, the interactions between the agent and environment are infinite; 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 we decrease the randomness when training. All transitions are saved into a memory buffer queue $\mathcal{D}$, the agent learns to optimize its action by minimizing the loss function L on a minibatch from $\mathcal{D}$. The training procedure will converge as 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, we list the parameters in Sec. 3.

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

Initialize replay memory queue $\mathcal{D}$ 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 random minibatch of transitions $(\phi_j, c_j, r_j, \phi_{j+1})$ from memory buffer $\mathcal{D}$
    Set $y_i = r_j + \gamma \max_{c'} Q(\phi_{j+1}, c'; \theta)$
    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 [28]
  end if
end for

2.4 Feedback reward 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 $\mathcal{A}_c$ in compression quality c. Basically, we want the agent to choose a proper compression level that minimizing the file 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 $R(\Delta s, \mathcal{A})$.

2.5 Inference-estimate-retrain mechanism

As a running system, we introduce a running-estimate-retrain mechanism to cope with the scenery change in the inference phase, building a system with different components to inference, capturing scenery change, then retraining the RL agent. The overall system diagram is illustrated in Figure 3.

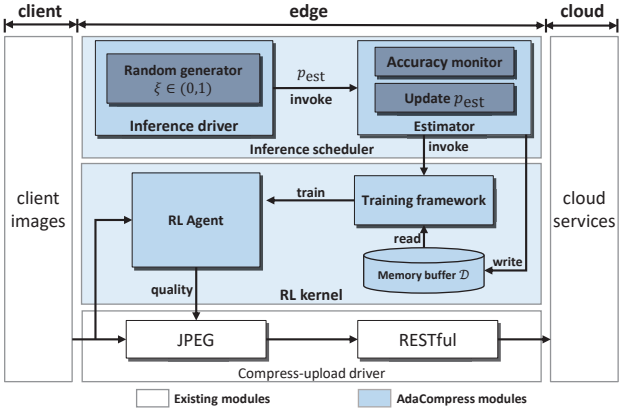


Figure 3: Diagram of AdaCompress architecture

The system diagram is shown in Figure 3. We build up the memory buffer $\mathcal{D}$ and RL (reinforcement learning) training kernel based on the compression and upload driver. When the RL kernel is called, it will load transitions from the memory buffer $\mathcal{D}$ to train the compression level predictor ϕ. When the system is deployed, the pre-trained RL agent ϕ guides the compression driver to compress the input image with an adaptive compression quality c according to the input image, then uploads the compressed image to cloud.

After the AdaCompress is deployed, the input images 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 this scenery drifting issue, we invoke an estimator with probability p_{est}. We do this by generating a random value $\xi \in (0, 1)$ and compare it to p_{est}. If $\xi \leq p_{\text{est}}$ the estimator is invoked, AdaCompress will upload the reference image x_{ref} along with the compressed image x_i to fetch $\vec{y}_{\text{ref}}$ and $\vec{y}_i$ and therefore calculates $\mathcal{A}_i$, and save the transition $(\phi_i, c_i, r_i, \mathcal{A}_i)$ to the memory buffer $\mathcal{D}$. The estimator will also compare the recent n steps' average accuracy $\bar{\mathcal{A}}_n$ and the earliest average accuracy $\mathcal{A}_0$ in memory

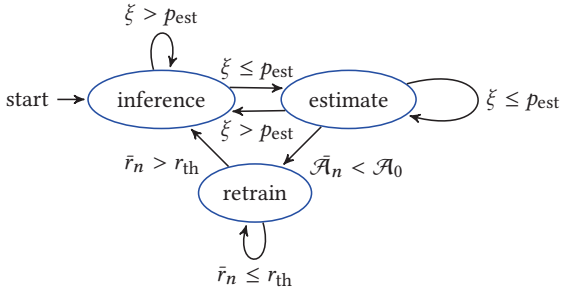


Figure 4: State switching policy

$\mathcal{D}$, once the recent average accuracy is much lower than the initial average accuracy, the estimator will invoke the RL training kernel to retrain the agent. And once the estimator discover that the trained reward is higher than a threshold, it will stop the training kernel, returning to normal inference state.

Basically, AdaCompress will adaptively switch itself between three states. The switching policy is shown as Figure 4.

2.5.1 ***Inference:*** For most times, AdaCompress runs in this state. In this state, only the compressed images are uploaded to the cloud to achieve minimum uploading traffic load. To keep a stable accuracy performance even the input scenery changes, the agent will occasionally switch to estimation state with probability p_{est}, meanwhile remains inference state with probability $1 - p_{\text{est}}$.

2.5.2 ***Estimate:*** In this state, the reference image x_{ref} and compressed image x_i are uploaded to the cloud simultaneously to fetch $\vec{y}_{\text{ref}}$ and $\vec{y}_i$ and therefore $\mathcal{A}_i$. In each epoch i the transition $(\phi_i, c_i, r_i, \mathcal{A}_i)$ is logged in a memory buffer $\mathcal{D}$. Once the average accuracy $\bar{\mathcal{A}}_n$ of the latest n steps is lower than the average accuracy $\mathcal{A}_0$ of the earliest n steps in the memory buffer $\mathcal{D}$, indicating that the current agent is no more suitable for the current input scenery, AdaCompress will switch into retrain state and invoke the RL training kernel. Otherwise, it remains estimate state with probability p_{est} or switches back into inference state with probability $1 - p_{\text{est}}$.

Therefore, the estimating probability 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, so that to retrain the agent on time once the scenery changes; on the other hand, the estimator will upload the reference image x_{ref} along with the compressed image, therefore the upload size is greater than the conventional benchmark solution, causing higher traffic load.

To trade-off between the risk of scenery changes and the objective of reducing upload traffic, 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:

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

With this definition, an intuitive formulation of the changes of p_{est} is in inverse proportion of the gradient of $\mathcal{A}$, meaning that when the recent accuracy is going down, we should increase the estimation probability p_{est}. 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_{est} with an initial estimation probability p_0 is $p_{\text{est}} = p_0 + \omega\sum_{i=0}^{N}\nabla\bar{\mathcal{A}}_i$.

2.5.3 ***Retrain:*** This state is to adapt the agent to the current input image scenery by retraining it with the memory buffer $\mathcal{D}$, which is similar to the training procedure. The retrain phase finishes upon the recent n steps' average reward $\bar{r}_n$ higher than a user-defined threshold r_{th}. And when the retrain procedure finishes, the memory buffer $\mathcal{D}$ will be flushed, preparing to save new transitions for the retraining of a next scenery drift.

2.6 Insight of RL agent's behavior

In the inference phase, the pre-trained RL agent predicts a proper compression 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 is reduced. We noticed that the RL agent's behavior are various for different input dataset and backend cloud services, we try to take further investigations by plotting the RL agent's "attention map" (i.e., visual explanations of why the agent chooses a quality level).

2.6.1 ***Compression level choice variation:*** In our experiment, we found that in different cloud application environments, the agent's final chosen compression qualities 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 model in the training phase. Since the agent's training procedure is based on a specific backend cloud model M_1, for another cloud model M_2, the interaction between the agent and M_2 is quite different. Therefore the agent's compression level choice presents variation for different backend cloud models.

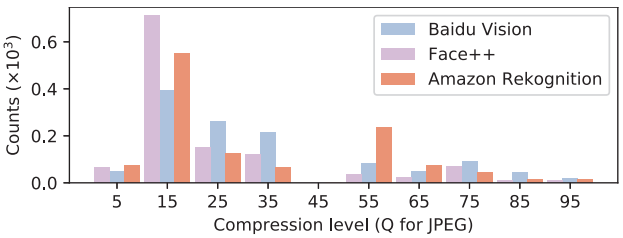


Figure 5: Histogram of RL agent's best compression level selection for different cloud services

Moreover, in our experiment, the agent presents different behavior when the input images change from one dataset to another. Figure 6 shows the agent's choices for a same backend model (Baidu Vision) but different image datasets. We prepare two datasets indicating two contextual scenery. We randomly sample images from ImageNet [33] whose images are mostly taken in the daytime, to act as a daytime scenery, and we randomly select nighttime images from DNIM [44] 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 level, but its

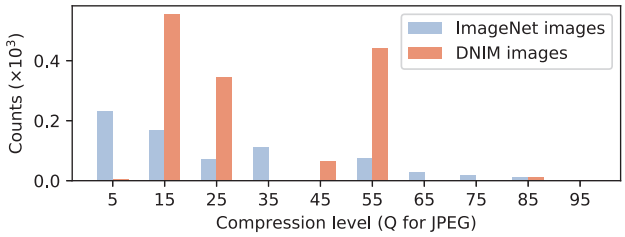


Figure 6: Histogram of RL agent's best compression level selection for different scenery image inputs

choices are distributed more evenly. For DNIM images, the agent's choices are more accumulated in some relatively high compression qualities. We can see that, to maintain high accuracy, when the input image's contextual group $\mathcal{X}$ changes, the agent's compression level selection changes as well. This phenomenon presents that the agent can adaptively choose a proper compression level based on the input image's features.

2.6.2 ***Attention map variation:*** To take insight investigation, we plot the importance map of a chosen compression quality. We do so by introducing a conventional visualize algorithm, Grad-Cam, to observe the Q prediction network's interest when choosing compression 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 investigation, we found that in different environment $\{\mathcal{X}, M\}$, the Q agent picks up compression qualities based on the visual textures of different regions in the image. As shown in Figure 7, picture 1a – 1d are some pictures that the agent chooses to compress highly, the agent selects lower compression qualities based on the complex texture of the images. On the contrary, for pictures 2a – 2d, the agent chooses higher compression qualities to preserve more details, and the agent's interest falls on some smooth regions. Especially for 1a and 2a, in picture 1a, the agent chooses a low compression level based on the rough central region though there are smooth regions around it, and in picture 2a, the agent chooses a relatively higher compression level based on the surrounding smooth region rather than the central region.

3 EVALUATION

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

3.1 Experiment setup

We carry out real-world experiments to verify our solution's performance. We used 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 service, and Amazon Rekognition. In the experiments, we use two datasets mentioned before in Sec.2.6, ImageNet dataset indicating daytime scenery and DNIM dataset indicating nighttime scenery. Some important hyperparameters in our experiments are given in Table 1.

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

Table 1: Experiment parameter settings

3.2 Metrics

In industry, the default compression quality for JPEG is usually 75 [26, 31], we regard this as a typical value $c_{\text{ref}} = 75$ of the conventional industry 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 from a reference image $\vec{y}_{\text{ref}}$ 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. 2.1.

3.3 Upload size overhead

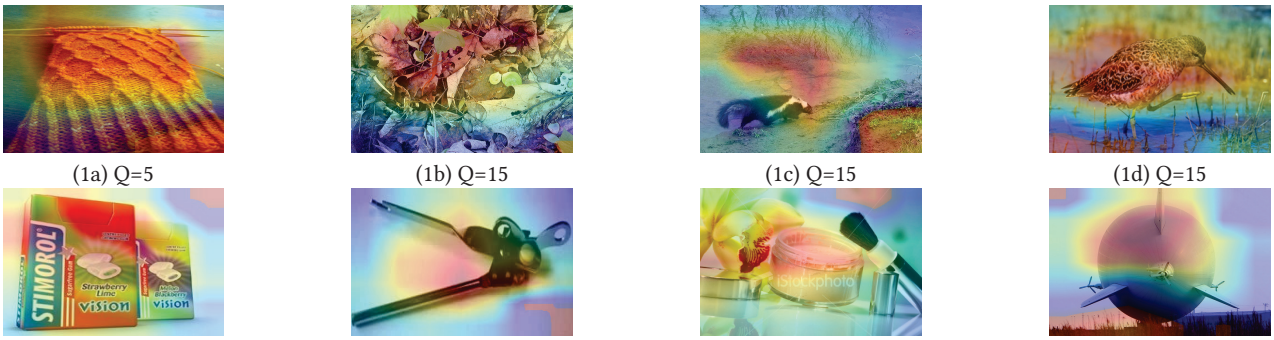
Figure 8 presents the upload traffic load of the training and inference phase, to be more intuitionistic, we plot the size overhead $\frac{s}{s_{\text{ref}}}$ as the y-axis where s is the real upload size of AdaCompress, s_{ref} is the benchmark upload size, therefore $y \geq 1$ means that our solution 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 uploaded size is decreasing because the DRL agent is learning to choose better quality levels to upload less 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 finished, the upload traffic is much 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.

3.4 Size reduction and accuracy performance

Figure 9 presents the compression performance in the inference phase for each cloud service. We tested 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 decrease about 7% on average, proving the efficiency of our design.

3.5 Adaptive retrain upon scenery change

To evaluate the efficiency of the inference-estimate-retrain mechanism, we feed AdaCompress with a combined dataset whose first 720 images from DNIM night images, the later 2376 images randomly sampled from ImageNet. We adapt AdaCompress's current DRL agent to DNIM night scenery by training it on DNIM dataset, then we run AdaCompress on the combine dataset, observing AdaCompress's behavior upon the scenery change at step 720.



(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

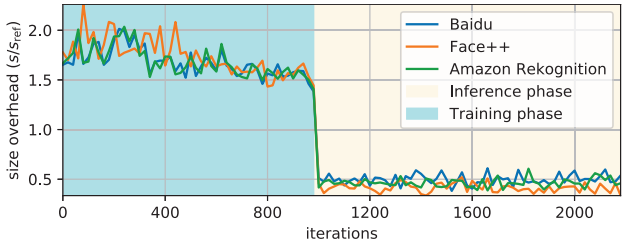


Figure 8: Size overhead in training and inference phase

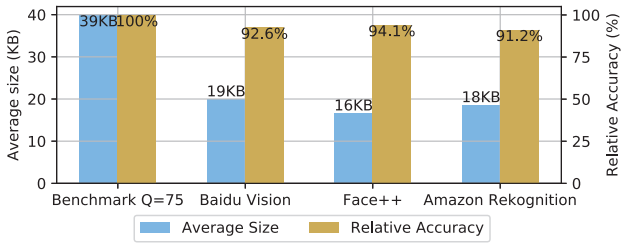


Figure 9: Average size and relative accuracy on different cloud services

We illustrate AdaCompress's behavior in Figure 10, the x-axis indicates steps, the vertical red line with a Δ mark on x-axis means the dataset change(i.e. scenery change). We plot AdaCompress's overall accuracy as the green line and the estimating probability p_{est} as the gray line. At the bottom of Figure 10, we also plot the scaled uploading data size of AdaCompress and benchmark solution to illustrate the upload data size overhead in the inference phase.

From Figure 10 we can see that AdaCompress can adaptively update the estimation probability p_{est}, usually, when the overall accuracy decreases, AdaCompress will increase the estimation probability, trying to catch the scenery change. When the overall accuracy is stable and high enough, the estimation probability p_{est} decreases to reduce transmission.

Upon the data scenery change shown as the vertical red line in Figure 10, comparing to the earlier steps, the accuracy decreases dramatically and therefore p_{est} raises to determine whether scenery changes, the accuracy keeps dropping in the following estimations.

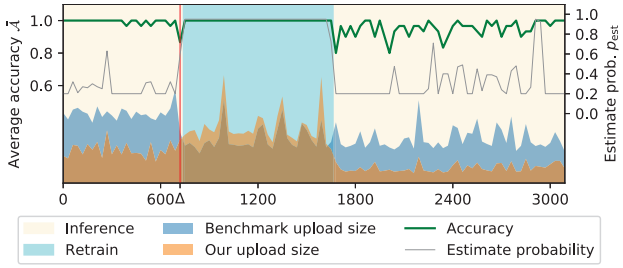


Figure 10: AdaCompress's reaction upon scenery change

Therefore, AdaCompress starts to retrain, to adapt the RL agent into the current scenery. The retrain steps are shown as the light-blue region in Figure 10. In the retrain phase, AdaCompress always uses the reference image's prediction label $\vec{y}_{\text{ref}}$ as the output result, therefore the accuracy $\mathcal{A}$ and p_{est} is locked to 1. After finishing retraining the agent in the new scenery, in the following iterations, sometimes the accuracy decrease accidentally, the estimation probability p_{est} also raises to get more samples, but the accuracy is not lower than the initial average accuracy $\mathcal{A}_0$ of this scenery, therefore the retrain phase will not be triggered again.

From Figure 10 we can also observe the uploading file size overhead in different phases, we can see that in retrain phase, AdaCompress uploads more data than the conventional benchmark, but in inference phase, AdaCompress's upload data size is only half of the benchmark's.

3.6 End-to-end latency simulation

Comparing to the conventional solution that uploads the image directly, in our solution, the image is passed to the DRL agent first to estimate the compression level. Running this DRL agent brings extra latency to the whole system. In this subsection, we evaluate this latency overhead.

We tested the DRL 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 from 1000 ImageNet images and simulate the network bandwidth as 27.64 Mbps according to the global average fixed broadband

upload speed [38] in Feb. 2019 to verify the end-to-end latency performance. The latency comparison is listed in Table 2.

	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 2: 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 [20, 35], we can use the computing power of the edge infrastructure in exchange for the reduction of upload traffic and transmission latency.

4 RELATED WORKS

As cloud-based computer vision services have become the norm for today's applications [1, 22], many studies have been devoted to improving the cloud-based model execution, including model compression and data compression.

4.1 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. A number of studies tried to compress the deep learning models and deploy them *locally* [3, 4, 13, 16, 17, 23], i.e., running an alternative "smaller version" of a computer vision model at the user end, to avoid the image upload, so that to improve the inference efficiency. Other studies proposed to run part of a deep learning model locally [9, 19, 24, 25], 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 cloud computer vision services that are merely a black box to end-users, e.g., in the form of a RESTful API.

4.2 Data compression

Data compression solutions study how to compress the original data (e.g., a video or image) to be inferred by the cloud deep learning model, so that less traffic is used to upload the data to improve inference speed. Conventional data compression solutions (e.g., JPEG, WebP, JPEG2000 etc.) and some recent neural network based compression solutions [32, 39–41] are initially designed for human vision systems. In recent years, researchers start to found that the human visually optimized data compression solutions are not usually applicable to deep learning vision systems. Delac et al. [7] observed that, in some cases, higher compression level does not always deteriorate the model inference accuracy, and in some cases, even improves it slightly. Dodge et al. [8] further discovered that besides the JPEG compression, four types of quality distortions: blur, noise, contrast, and JPEG2000 compression can also affect the performance in deep learning inference.

Based on these insights, Robert et al. [42] tried to train the neural network from the compressed representations of an auto-encoder. Liu et al. [27] proposed DeepN-JPEG that provides a JPEG quantization table learned from the dataset so that the compressed image size is reduced for deep learning models. Recently, Lionel et al. [15] present a new type of neural network that inference directly from the discrete cosine-transform (DCT) 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 the image.

However, such proposals all need one to understand the characteristics of the cloud-end 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 that learns the deep learning model by itself.

5 CONCLUSION AND FUTURE WORK

To reduce the upload traffic load of deep learning application, 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 a deep learning agent to decide the proper compression quality for each image, according to the input image and backend service. Our experiments show that for different backend deep learning cloud services and different input image scenery, using different quality selection strategy can significantly reduce the upload file size while keeping comparable accuracy. Based on this work, some possible future orientations can focus on the following: 1) In some regularly change scenery (e.g., daytime and nighttime, etc.), one can design an agent caching strategy, to cache an agent for a specific scenery and use it again when a similar scenery arrives rather than retrain from scratch. 2) By introducing transfer learning and knowledge distillation, an agent could learn from another nearby agent to accelerate its training.

ACKNOWLEDGMENTS

This work is supported in part by NSFC under Grant 61872215, 61531006, 61771273 and U1611461, National Key R&D Program of China under Grant 2018YFB1800204 and 2015CB352300, SZSTI under Grant JCYJ20180306174057899 and JCYJ20180508152204044, and Shenzhen Nanshan District Ling-Hang Team under Grant LHTD20170005.

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. (2019).

[7] 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.

[8] 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.

[9] 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.

[10] 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*.

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

[12] 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.

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

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

[15] 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.

[16] 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).

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] Huawei. 2019. Huawei Atlas 500 Edge Station. https://e.huawei.com/en/products/cloud-computing-dc/servers/g-series/atlas-500.

[22] 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.

[23] 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.

[24] 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. ACM, 615–629.

[25] 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

[26] Python Imaging Library. 2019. Image file formats. https://pillow.readthedocs.io/en/3.1.x/handbook/image-file-formats.html.

[27] 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.

[28] 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).

[29] Jia D et al. Olga R. 2012. ImageNet Large Scale Visual Recognition Challenge 2012. http://image-net.org/challenges/LSVRC/2012/.

[30] 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).

[31] rflynn. 2019. Lossy image optimization. https://github.com/rflynn/imgmin.

[32] 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.

[33] 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

[34] 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.

[35] Mahadev Satyanarayanan. 2017. The emergence of edge computing. *Computer* 50, 1 (2017), 30–39.

[36] 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.

[37] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[38] SpeedTest. 2019. Speedtest Global Index. https://www.speedtest.net/global-index.

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

[40] 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).

[41] 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.

[42] 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).

[43] 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).

[44] Hao Zhou, Torsten Sattler, and David W Jacobs. 2016. Evaluating local features for day-night matching. In *European Conference on Computer Vision*. Springer, 724–736.