Genie in the Model: Automatic Generation of Human-in-the-Loop Deep Neural Networks for Mobile Applications

YANFEI WANG, Northwestern Polytechnical University, China
ZHIWEN YU, Northwestern Polytechnical University, China
SICONG LIU, Northwestern Polytechnical University, China
ZIMU ZHOU, City University of Hong Kong, China
BIN GUO, Northwestern Polytechnical University, China

Advances in deep neural networks (DNNs) have fostered a wide spectrum of intelligent mobile applications ranging from voice assistants on smartphones to augmented reality with smart-glasses. To deliver high-quality services, these DNNs should operate on resource-constrained mobile platforms and yield consistent performance in open environments. However, DNNs are notoriously resource-intensive, and often suffer from performance degradation in real-world deployments. Existing research strives to optimize the resource-performance trade-off of DNNs by compressing the model without notably compromising its inference accuracy. Accordingly, the accuracy of these compressed DNNs is bounded by the original ones, leading to more severe accuracy drop in challenging yet common scenarios such as low-resolution, small-size, and motion-blur. In this paper, we propose to push forward the frontiers of the DNN performance-resource trade-off by introducing human intelligence as a new design dimension. To this end, we explore human-in-the-loop DNNs (H-DNNs) and their automatic performance-resource optimization. We present H-Gen, an automatic H-DNN compression framework that incorporates human participation as a new hyperparameter for accurate and efficient DNN generation. It involves novel hyperparameter formulation, metric calculation, and search strategy in the context of automatic H-DNN generation. We also propose human participation mechanisms for three common DNN architectures to showcase the feasibility of H-Gen. Extensive experiments on twelve categories of challenging samples with three common DNN structures demonstrate the superiority of H-Gen in terms of the overall trade-off between performance (accuracy, latency), and resource (storage, energy, human labour).

CCS Concepts: • **Human-centered computing → Ubiquitous and mobile computing systems and tools**.

Additional Key Words and Phrases: Human in the Loop, neural networks, model generation, reinforcement Learning

ACM Reference Format:
Yanfei Wang, Zhiwen Yu, Sicong Liu, Zimu Zhou, and Bin Guo. 2023. Genie in the Model: Automatic Generation of Human-in-the-Loop Deep Neural Networks for Mobile Applications. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 7, 1, Article 36 (March 2023), 29 pages. https://doi.org/10.1145/3580815

Corresponding author: Zhiwen Yu (zhiwenyu@nwpu.edu.cn).
Authors' addresses: Yanfei Wang, Northwestern Polytechnical University, School of Computer Science, Xi'an, China; Zhiwen Yu, Northwestern Polytechnical University, School of Computer Science, Xi'an, China ; Sicong Liu, Northwestern Polytechnical University, School of Computer Science, Xi'an, China; Zimu Zhou, City University of Hong Kong, School of Data Science, Hong Kong, China; Bin Guo, Northwestern Polytechnical University, School of Computer Science, Xi'an, China.

2474-9567/2023/3-ART36 $15.00
https://doi.org/10.1145/3580815

1 INTRODUCTION

There is a growing interest to deploy deep neural networks (DNNs) on mobile and embedded devices for intelligent applications and services [1, 27, 44, 50]. Examples include user authentication on smartphones [54], voice assistant on smartphones [65], facial action recognition on smart eyewear [60], and activity recognition on Smart-Home devices [5]. However, deploying DNNs in mobile applications faces two critical challenges. On the one hand, modern DNNs are often resource-hungry, which easily overwhelm the storage, computation, and energy of mobile devices. For instance, Faster RCNN [49] which is widely used for fast object detection, still requires $372MB$ storage and $1826M$ MACs per inference. This is unacceptable for resource-scarce mobiles and wearables such as smart-bands and smart-watches. On the other hand, benchmark-trained DNNs often incur accuracy degradation when deployed in real-world mobile scenarios, particularly in challenging environments. For example, the noisy environment (*e.g.* low-light), the sensor limitation (*e.g.* low resolution), and some shooting context (*e.g.* small, motion-blur, or camouflaged objects) often lead to challenging input samples and result in severe accuracy degradation. These challenging scenarios are common in real-world mobile applications.

To improve the resource efficiency of DNNs, various model compression techniques have been proposed to reduce the resource demand of DNNs without significantly compromising their accuracy [23, 53, 66]. Standalone model compression techniques include pruning [9, 25, 33], quantization [42], knowledge distillation [24]. Some research [38] also developed automatic selection frameworks to combine multiple compression techniques to achieve higher accuracy and compression ratio. Despite extensive research on model compression, the compressed DNNs typically yield lower accuracy than the original model. Therefore, they still suffer the same, if not worse, performance degradation as the original DNNs in challenging scenarios.

To push the limit of the DNN's performance-resource trade-off, we advocate human intelligence as a new dimension in DNN optimization. Multiple reports [13, 31, 64] observe that humans are more robust and reliable than DNNs in certain cases. For instance, one study [13] empirically showed that humans outperform DNNs on classification tasks for distorted stimuli (blur or noise distortions). Another study [31] demonstrated through the Turing test that the human vision can detect uncommon, camouflaged or obscured objects more effectively than object detection DNNs. These observations have stimulated the exploration on human-in-the-loop machine learning [7], where human knowledge and experience is integrated into the development of machine learning models for higher accuracy or faster workflow [59, 62, 69]. However, existing human-in-the-loop machine learning proposals [4, 8, 74] are inapplicable to optimize the performance-resource trade-off in DNN design. This is because they mainly focus on human participation mechanisms *e.g.* data annotation for dedicated learning models or paradigms, without accounting for the resource constraints or the cost of human labour.

In this paper, we propose H-Gen, an automatic model compression framework for human-in-the-loop DNNs (H-DNNs). The idea is to break the accuracy limit of the original DNN with human intelligence while formally modeling the impact of human participation on the performance and resource utilization of DNNs. From a high level, H-Gen incorporates human participation as a new dimension to optimize the performance (*e.g.* accuracy, latency) and resource (*e.g.* storage, energy) of common DNN architectures in mobile applications. Like a genie in the bottle, we consider human participation as the genie in the (deep) model, and quantify both its gains and cost into the optimization of DNNs. The design and implementation of H-Gen faces two technical challenges.

- *How to integrate human participation into DNNs,* i.e.*, H-DNNs.* Following the concepts of human-in-the-loop machine learning, we design H-DNNs as DNNs with opportunistic human participation to guide the DNN inference on uncertain input samples. In line with prior research [4, 72], we adopt human annotation as the participation mechanisms and implement them as crowdsourcing tasks. We design dedicated human annotation mechanisms for three DNN architectures commonly seen in mobile vision applications.
- *How to model human participation to optimize the performance-resource trade-off of DNNs.* Although human participation improves model accuracy, it also incurs extra monetary cost and inference latency due to the

involvement of crowdsourcing-based human annotation. We quantify the impact of human participation as a unified hyperparameter and propose an automatic hyperparameter optimization framework to trade-off the performance requirements and resource constraints of H-DNNs.

We implement H-Gen's server side in Python and mobile side (application) in JAVA, and evaluate its performance over three H-DNN architectures on twelve challenging datasets with three different mobile devices. Evaluations show that the H-DNNs generated by H-Gen can improve accuracy by 0.6% $\sim$ 7.3% and reduce parameter size by 1.9× $\sim$ 18.6×, latency by 0.8× $\sim$ 2.9×, energy cost by 1.1× $\sim$ 2.8× with labour cost 0.6% $\sim$ 6.8% compared with original DNNs.

The main contributions of this work are summarized as follows.

- Conceptually, this is the first work that incorporates human participation as a new optimization dimension into the design space of efficient DNNs for better performance-resource trade-offs on mobile applications.
- Technically, we model human participation as a hyperparameter and devise H-Gen, an automatic hyperparameter tuning framework that generates H-DNNs with high performance (accuracy, latency) and low resource overhead (storage, energy, and human labour). We also propose human participation mechanisms for three common DNN architectures to showcase the feasibility of H-Gen.
- Extensive experiments show that H-Gen achieves higher inference accuracy, lower energy cost, and smaller storage footprint with little labour cost for various challenging tasks and mobile platforms.

In the rest of the paper, we review the related work in Section 2, present an overview of H-Gen in Section 3, elaborate on the two components in Section 4 and Section 5, and explain its implementation in Section 6. We report the evaluation of H-Gen in Section 7, and finally conclude in Section 8.

2 RELATED WORK

2.1 Human-in-the-Loop Machine Learning

Human-in-the-loop (HIL) machine learning [7] is a human-machine computing paradigm that integrates human knowledge and experience into the development of machine learning models [59, 69]. It aims to improve the model accuracy [59], accelerate the model development workflow [62], or provide better responsibility [70]. We focus on improving model accuracy by exploiting human intelligence to handle difficult samples in DNN-based mobile applications. Annotation is the most widely adopted type of human involvement, where humans are asked to annotate samples or intermediate results [2, 28, 41, 73], although humans may also participate in data preparation such as sample selection and normalization [18, 21, 78]. Here the primary design challenge is *when* and *how* to integrate human annotation into the machine learning loop, which varies across learning paradigms and applications. Prior work mainly deals with two learning paradigms: supervised learning [4, 64, 72] and reinforcement learning [58, 74, 75]. For example, Yang *et al.* [64] proposed a HIL video anomaly detection model, where human experts judge whether there is an abnormality in the video frame, and annotate either true or false to the model. Zhang *et al.* [72] developed an interactive disaster scene assessment scheme with human annotation for accuracy improvement, by crowd-sourcing the annotations of the salient regions in disaster images with low classification confidence. Arous *et al.* [4] designed a HIL model to find social influencers by annotating wether a few candidates are critical social influencers by human experts. Zhang *et al.* [75] summarized different ways (*e.g.* learning from human evaluation feedback, learning from human preference) to combine human knowledge and decision-making with reinforcement learning to improve its effectiveness. Liu *et al.* [40] proposed a deep reinforcement active learning method to corporate human intelligence into the pedestrian re-identification loop through iterative labeling.

We *make the first attempt* at generating human-in-the-loop DNNs for mobile applications. Unlike prior HIL supervised learning research [4, 64, 72] that merely considers the *gains* of human participation, we also account for the *cost* of human participation, and incorporate model accuracy, human cost, and mobile devices' resource

constraints into an automatic optimization framework. Moreover, our work's goals differ from the active learning methods. Our work is *orthogonal to* existing HIL reinforcement learning studies because we apply standard reinforcement learning to optimize the hyperparameters of human-in-the-loop DNNs rather than involve humans in the process of reinforcement learning.

2.2 Neural Architecture Search

Neural Architecture Search (NAS) automates the architecture engineering of neural networks [16], and has generated models with accuracy that outperforms manually designed ones in many tasks. A typical NAS framework consists of three elements: search space, search strategy, and model performance evaluation. Since the search space of NAS is often exponential, extensive research strives to reduce the cost of NAS in various dimensions [35, 77, 79]. In addition to accuracy, NAS has also been applied to cope with resource constraints. For example, MnasNet [55] incorporates model latency into the objective to optimize both accuracy and latency. PPPNet [14] is an efficient NAS scheme to search for Pareto-optimal architectures under multiple objectives. ChamNet [11] devises accuracy and resource *e.g.* latency and energy predictors for multi-objective architecture search.

In addition to fine-grained architecture search, NAS is also widely used for high-level hyperparameter optimization. Zela *et al.* [71] combined network architecture and hyperparameter search efficiently and used a probabilistic model to sample promising configurations to optimize seven hyperparameters such as initial learning rate and batch size. Dong *et al.* [15] generalized the concept of efficient architecture search to hyperparameter search and adopted an RL controller to learn the probability distribution for the hyperparameter candidates.

In this paper, we adopt the NAS concepts for automatic hyperparameter optimization of human-in-the-loop DNNs (H-DNNs). Specifically, we define the search space of H-DNNs that incorporates human efforts as a new hyperparameter, and design a two-stage search strategy to generate effective and efficient models.

2.3 DNN Generation for Mobile Devices

Deploying DNNs to mobile devices faces unique challenges because DNNs easily exceed the resource limits of mobile devices. There have been various standalone model compression techniques to reduce the resource footprint of DNNs without notably compromising their accuracy, such as pruning [34], quantization [42], knowledge distillation [24] *etc.* Multiple model compression techniques can also be combined for better accuracy-resource trade-off. For example, Polino *et al.* [47] leveraged quantization and knowledge distillation to achieve similar accuracy to full-precision teacher models with up to order of magnitude compression. Liu *et al.* [37] proposed a context-based fast online DNN compression algorithm by adaptively selecting the elite compression operations.

Our work is most related to AdaDeep [36, 38], which automatically selects and combines different compression techniques to generate DNNs according to diverse task requirements and resource constraints. However, the DNNs generated by AdaDeep may incur notable accuracy loss under severe resource budgets because the accuracy is restricted by that of the uncompressed model. We *break this accuracy restriction by introducing human involvement*, which improves the accuracy of the uncompressed model in case of challenging input samples (as enumerated in Section 7.1.1, where the concrete evaluation results are in Section 7.2.2).

We also *model the cost* of human involvement into the model generation framework and achieve a better overall accuracy-resource trade-off.

3 H-GEN OVERVIEW

This section presents an overview of H-Gen, a framework to automatically generate human-in-the-loop DNNs that deliver better performance-resource trade-off than conventional DNNs. The key novelty of H-Gen is a NAS-style formulation and solution that incorporates human annotation as a new optimization dimension by modeling both its performance gain and resource cost for the DNNs. And our goal is to break the accuracy restriction

of previous automation frameworks such as AdaDeep [38] by introducing human involvement. We showcase H-Gen with three DNN architectures widely adopted in mobile vision applications, including CNN [30], YOLO [48], and Faster R-CNN [49]. We elaborate on the components and workflow of H-Gen below.

3.1 H-Gen Framework

Given a pre-defined human-in-the-loop DNN (H-DNN), H-Gen optimizes its hyperparameters (network architecture, degree of human participation) as well as its parameters (weights) to achieve high performance (*e.g.* accuracy) with low cost (both device resource utilization and human efforts). From a high level, H-Gen is an *automatic model generation scheme* that optimizes various *human-in-the-loop DNNs*. We briefly explain the design space of these two elements below.

- **H-DNNs.** As mentioned in Section 1, H-DNNs are motivated for improved accuracy by guiding the inference on uncertain and challenging input samples (*e.g.* noisy samples, low-light images and small objects) with human annotation. Given a DNN architecture, its corresponding H-DNN defines *how* and *when* to involve human annotation as well as the *supportive mechanism* for human participation. As with previous HIL machine learning research [72], we utilize commercial crowdsourcing platforms *e.g.* Amazon MTurk[3] as the mechanism to support human participation, which performs task generation, participant selection, and annotation fusion. How to integrate human annotation into DNNs is architecture-specific. In this paper, we propose feasible human annotation mechanisms for three common DNN architectures (see Section 4.1). When to trigger human annotation is determined by the mobile application demands on inference latency and human labour cost. This is because human crowdsourcing incurs monetary cost and extra latency on the inference task. We determine the cost of human annotation by a unified threshold τ, and assess its impact on inference latency via a statistical method (see Section 5.2).
- **Automatic H-DNN Generation.** Given a specific H-DNN architecture, H-Gen automatically optimizes its performance via a NAS-style hyperparameter optimization scheme in the training stage. H-Gen considers a combination of model compression techniques [38] and the degree of human annotation as the hyperparameter search space (see Section 5.1), calculates the corresponding performance and resource metrics (see Section 5.2), and takes a two-stage reinforcement learning based optimization strategy to generate an H-DNN (see Section 5.3). The objective is to optimize the H-DNN model's performance (high accuracy, low latency) under resource constraints (storage, energy, and human efforts). Then the generated H-DNN is loaded to conduct human-in-the-loop (HIL) inference for testing on mobile devices.

3.2 H-Gen Workflow

Figure 1 illustrates the workflow of H-Gen. For a given H-DNN architecture (DNN and human annotation mechanism), H-Gen takes user-specified performance demands (accuracy, latency) and resource constraints (platform's storage and energy budgets, human labour) as inputs and searches the degree of model compression and human annotation to the H-DNN. Such that these performance metrics are optimized under resource constraints.

Formally, H-Gen focuses on the following hyperparameter optimization problem.

$$\begin{aligned} \underset{\upsilon\in,\tau\in[0,1]}{argmax} \quad & \delta_1 log(A - A_d) + \delta_2 log(T_d - T) \\ \text{s.t.} \quad & S \le S_c,\ E \le E_c,\ L \le L_c \end{aligned} \tag{1}$$

where A, T, S, E, and L are the accuracy, latency, storage, energy, and the associated human labour cost of the generated H-DNN. S and E are gathered in advance, and A, T, while L are measured based on a batch of test data (see Section 5.2). A_d and T_d are user-specified performance requirements on accuracy and latency, while S_c, E_c and L_c denote the resource constraints on storage, energy, and human labour. These bounds are predefined by users as inputs for H-Gen. H-Gen optimizes A, T, S, E, and L by tuning the hyperparameters of the H-DNN,

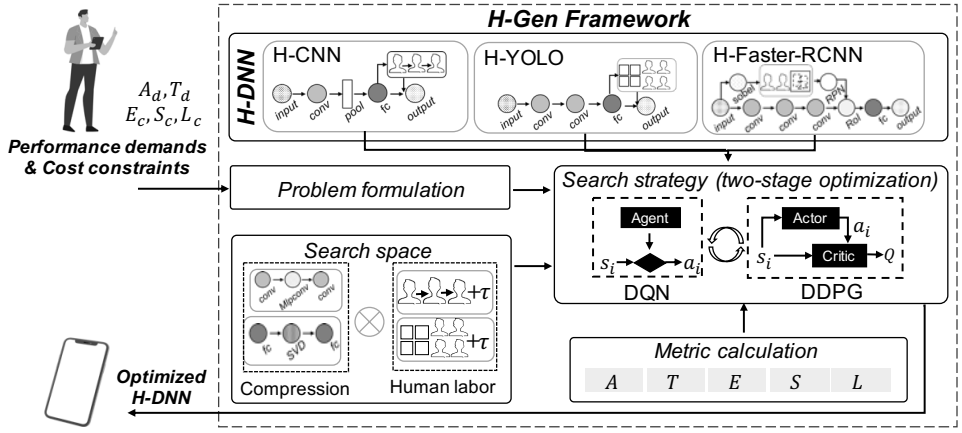


Fig. 1. An overview of H-Gen, which automatically generates H-DNNs with optimized performance-resource trade-off.

which includes υ, model compression on the H-DNN backbone, and τ, which determines the amount of human annotation. We use Φ to represent the corresponding search space for model compression. The search space for human participation hyperparameter is set as $\tau \in [0, 1]$ (see Section 5.1 for details). δ_1 and δ_2 are coefficients to balance A and T and we use $log(.)$ for normalization, *i.e.*, converting them to the same scale [37].

The H-DNN generated by H-Gen framework is then ready to deploy for inference. When deployed on mobile applications for inference, the H-DNN will assess the confidence of the inference on each input sample, and human annotation will assist in the inference on samples with low confidence (determined by the threshold τ). Specifically, these low confidence samples will be sent to the crowdsourcing platform for human annotation, and then the feedback is sent to the model to complete the final inference. The human annotations will also be stored in an extra human knowledge base to update the H-DNNs continuously. We discuss the concrete implementation and usage of H-DNNs in Section 4.2.

Discussions. We make the following notes on the problem formulation of H-Gen.

- We formulate the hyperparameter tuning problem by optimizing accuracy and latency, with storage, energy and human participation as constraints because accuracy and latency are closely coupled with the quality of service in mobile applications [14, 36, 38]. Note that human participation in HIL machine learning is typically opportunistic [7]. Therefore, we restrict the human labour cost for a batch of inferences, *i.e.*, the number of annotations should not exceed a predefined budget for a given batch of inference tasks.
- We advance prior DNN hyperparameter tuning formulations [36, 38] by involving human participation as a new optimization dimension. It is a non-trivial formulation because human annotation brings improved accuracy at the cost of extra latency and monetary overhead. Accordingly, we need novel designs on *(i)* quantifying the gains and cost of human efforts for diverse H-DNNs; and *(ii)* searching strategies for human efforts as a new hyperparameter.

4 H-DNN DESIGN

This section introduces the architectures of H-DNNs (Section 4.1) as well as their usage during inference and training (Section 4.2). These H-DNNs are the inputs for H-Gen for further performance and resource optimization. Note that our focus is on the automatic optimization of H-DNNs rather than the most user-friendly way for human participation. Thus, in this work, we only propose feasible human annotation mechanisms for three typical DNN architectures for mobile applications.

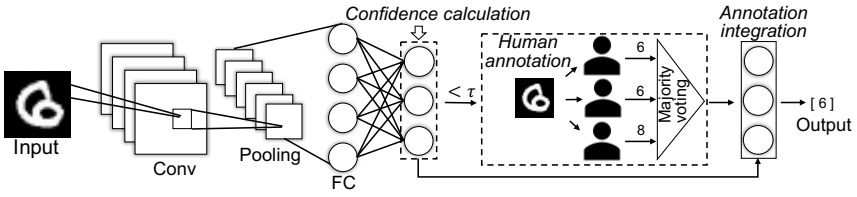


Fig. 2. An illustration of H-CNN, where humans annotate the category of images, and the annotations are aggregated by majority voting (see Section 4.1.1).

4.1 H-DNN Architectures

As mentioned in Section 3.1, H-DNNs are DNNs that involve human annotation on uncertain input samples with low inference confidence to improve the model accuracy. Accordingly, there are three questions when designing the H-DNN for a given DNN architecture: *(i)* how to assess the inference confidence (*i.e.*, uncertainty) of an input sample to trigger human annotation; *(ii)* what immediate results of the DNN model to annotate; and *(iii)* how to integrate the human annotation into the DNN output. We answer these questions by demonstrating feasible H-DNN designs for three DNN architectures, CNN [30], YOLO [48], and Faster R-CNN [49], as explained below.

4.1.1 H-CNN. Convolutional neural networks (CNNs) such as LeNet [30] and VGG [52] are widely used for various classification tasks (*e.g.* image, human activity, and acoustic event). A typical CNN consists of convolutional, pooling, and fully connected layers. Given a CNN for image classification on k categories, we propose the human-in-the-loop design of CNNs, denoted as H-CNNs, by incorporating annotating the true classification category of the input samples with low inference confidence (see Figure 2).

- *Confidence Calculation.* We trigger human annotation in H-CNNs using the softmax output of the last layer (typically a fully connected layer) as the confidence metric. The probability of the output category does not directly represent the confidence of the result, so it needs to be calibrated by Temperature Scaling [20] during the validation phase of the model. The confidence of the final H-CNN is defined as:

$$c_{H-CNN} = max_k \sigma_{SM}(z, k/\mathcal{T}) \tag{2}$$

  where c_{H-CNN} represents the confidence of the H-CNN inference for a sample. max_k means the largest value in a k-dimensional vector. z is the logit k-dimensional vector of the sample, *i.e.*, the input of the softmax layer of the model. σ_{SM} is the softmax operator. $\mathcal{T}$ is a learnable parameter by optimizing the cross-entropy loss function of the validation set samples.
- *Human Annotation.* H-CNN uses a threshold τ to determine the timing of human participation. When the confidence c_{H-CNN} is lower than the threshold τ, H-CNN will revise and update the result by human majority voting mechanism. Specifically, H-CNN sends the sample image to crowdsourcing platform and m human participants will be selected to annotate the category of this image. Each annotation will be mapped to a k-dimensional one-hot format vector during feedback, recorded as $anno_i$. We assume that the human annotators are always available and ready to annotate the data timely with high quality when requested because the crowdsourcing platform will check the participant's status and only assign the annotation tasks to the participants available.
- *Annotation Integration.* We sum the m human annotation vectors to update the output of the H-CNN softmax layer. Specifically, the softmax output vector of the H-CNN model is updated to $\sigma_{SM}(z, k/\mathcal{T}) + \lambda \sum_{i=1}^{m} anno_i$. λ is an adjustment factor (default as 0.1).

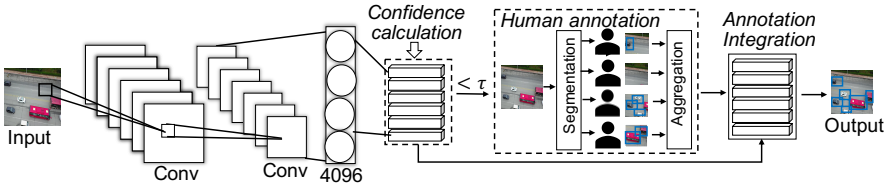


Fig. 3. An illustration of H-YOLO, where humans annotate the objects that exist in each image segment, and the annotations are aggregated by performing their union (see Section 4.1.2).

The cost per annotation is calculated by the task annotation difficulty d_{H-CNN} and the number of operations m (each annotation needs only one operation in single-classification tasks). Here we utilize an estimation function [10] to predict the annotation difficulty of H-CNN. We assign 3000 samples with known ground truth for human annotation and measure the annotation error rates to train the estimation function offline. Hence, the human labor cost per sample L^0_{H-CNN} is accumulated as in Eq.(3) below.

$$L^0_{H-CNN} = d_{H-CNN} \times m \tag{3}$$

4.1.2 H-YOLO. YOLO is a one-stage object detection model that regards the object detection task as a regression problem of object region prediction and class prediction [48]. It divides the original image into an $G \times G$ grid. Each grid needs to predict different bounding boxes and the class probabilities, and at most, one object is predicted in each grid. A known flaw in YOLO is its low detection rates for small objects [61]. In contrast, human vision often recognize small objects better than neural networks even in case of disturbances [31]. We design H-YOLO to incorporate human efforts into YOLO's last layer (classification and regression layer) (see Figure 3).

- *Confidence Calculation.* We calculate the confidence c_{H-YOLO} of the detection intermediate results (output dimension of the classification and regression layer) for a given image. The lower the detection confidence in a grid, the higher the probability of missing small objects, and the smaller contribution to the overall confidence. Thereby, the confidence is defined as follows:

$$c_{H-YOLO} = \frac{1}{G \times G} \sum_{i=1}^{G \times G} \mathcal{I}(c_{c_i}, c_{r_i}) \tag{4}$$

  where c_{c_i} and c_{r_i} are the classification confidence and regression confidence in each grid i. $\mathcal{I}$ is an indicator function defined by Eq.(5). When c_{c_i} and c_{r_i} of a grid i are both below the threshold of 0.5, we set them as 0. The threshold of 0.5 is empirically set to achieve a reasonably high detection accuracy (see Section 7.4.6).

$$\mathcal{I} = \begin{cases} 0 & c_{c_i} < 0.5, c_{r_i} < 0.5 \\ 1 & otherwise \end{cases} \tag{5}$$

- *Human Annotation.* H-YOLO also uses a threshold τ to determine the timing of human participation. When the confidence c_{H-YOLO} is lower than the threshold τ, H-YOLO uses a human segmentation aggregation mechanism to update the intermediate results. Specifically, the input image is equally divided into m parts, and we recruit m participants from the crowdsourcing platform for annotation. We employ the fusion mechanism [56] if an object is divided and sent to two annotators. Each participant annotates the objects that exist in the segmented image part. Each annotation is recorded as a set $\{anno_1(w, h, x, y, c), ..., anno_n(w, h, x, y, c)\}$, where w, h, x, y and c are the size, position and class of the human annotated objects. n is the number of objects annotated by each participant.
- *Annotation Integration.* We perform the union of the annotation sets of the m participants and then map it to the vector format consistent with the output of the YOLO model, to update the output layer results.

Eq.(6) shows the calculation of the human labour cost per sample for H-YOLO. Here, we estimate the annotation difficulty d_{H-YOLO} using a similar function, as mentioned in Section 4.1.1.

$$L^0_{H-YOLO} = \sum_{1}^{m} d_{H-YOLO} \times n_j (j = 1, 2, , m) \tag{6}$$

Note the annotation bounding-box number in a grid $n_j (j = 1, 2, , m)$ varies across human participants because each grid may contain different numbers of objects.

4.1.3 H-Faster-RCNN. Faster R-CNN [49] is a two-stage object detection model, which proposes Region Proposal Network (RPN) based on fast R-CNN [19]. It first extracts the features of the image by a backbone and then obtains the feature map. In the first stage, the feature map is passed to the RPN network to traverse pixel by pixel using nine a priori anchors and generate the proposals by classification and regression convolutional layers. In the second stage, the feature map of the convolutional layer is fixed as the input dimension of the fully connected layer using ROI pooling. Finally, it maps the proposals output by RPN to the feature map of ROI pooling for final box regression and classification. One problem in faster R-CNN is that the prior anchors cannot well cover the multi-scale scene of objects, which are usually unevenly distributed [46]. Thus, we design the human-in-the-loop version, H-Faster-RCNN, by exploiting humans to generate the proposals (see Figure 4).

- *Confidence Calculation.* In H-Faster-RCNN, we estimate the probability distribution of each localization and classification head output using a hybrid density network [10], which explicitly exhibits the arbitrary and epistemic uncertainty in a single forward pass of the model. Eq.(7) shows the confidence calculation.

$$c_{H-Faster-RCNN} = \sum_{p=1}^{P} \pi^p \frac{e^{\mu_i^p}}{\sum_{j=0}^{k} e^{\mu_j^p}} \tag{7}$$

  where P is the number of components in the hybrid density network. The mixture weight π^p is the p-th component, and the μ^p is the mean of GMM.
- *Human Annotation.* When c_f is lower than the threshold τ, we devise a region merging mechanism to guide the inference of H-Faster-RCNN. There will be m participants from the crowdsourcing platform for annotation. Specifically, each participant annotates n attention areas of the input image where there are multiple objects of uneven scale. Each annotation is recorded as a set $\{anno_1(w, h, x, y), ..., anno_n(w, h, x, y)\}$.
- *Annotation Integration.* Based on the feedback of human annotation, H-Faster-RCNN builds a Gaussian mixture model (GMM) [67] to establish a probability density map of the possible locations of the target. The calculation of the probability density map GMM is shown by Eq.(8).

$$GMM = \sum_{i=1}^{\Omega} \mu_i g(W, H | \bar{w}, \bar{h}, \bar{x}, \bar{y}, \Sigma) \tag{8}$$

  where $g(\cdot)$ is the density function of each Gaussian component. μ_i represents the mixture weight for each Gaussian component. $\bar{w}$, $\bar{h}$, $\bar{x}$ and $\bar{y}$ represent the mean of the size and position of the human annotation areas, respectively. Σ is a covariance matrix of size and position of human annotation areas. The optimal component number Ω is determined by Akaike's Information Criterion [76]. At pixels with high probability in GMM, we set more scale prior anchors (1:3 and 3:1 for three different anchor sizes) to better detect objects of different scales.

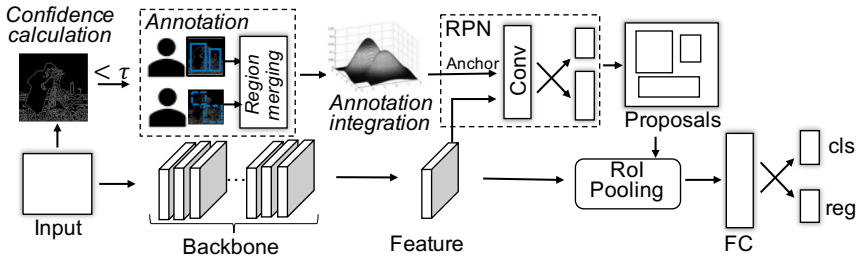
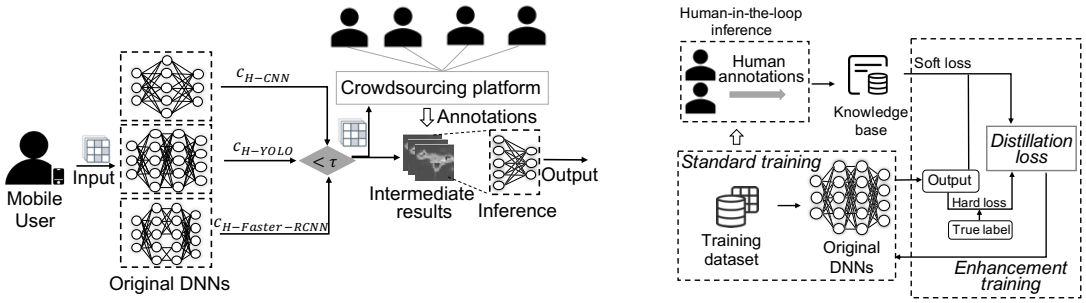


Fig. 4. An illustration of H-Faster-RCNN, where humans annotate attention areas of the input image, and the annotations are aggregated by region merging (see Section 4.1.3).



(a) H-DNN Inference (b) H-DNN Training

Fig. 5. Illustration of H-DNN in the (a) inference and (b) training phase.

The estimation method of $d_{H-Faster-RCNN}$ is similar to that in Section 4.1.1. The human annotation cost per sample is shown in Eq.(9).

$$L^0_{H-Faster-RCNN} = \sum_{1}^{m} d_{H-Faster-RCNN} \times n_j (j = 1, 2, , m) \tag{9}$$

Note that $n_j (j = 1, 2, , m)$ is different for each human because each participant focuses on different areas.

Note that we showcase these three human-in-the-loop deep models (*i.e.*,, H-CNN, H-YOLO, H-Faster RCNN) to verify the effectiveness of our H-Gen framework. Designing optimal structures of these three specific models is not the primary focus of our work.

4.2 H-DNN Inference and Training

We now briefly discuss how to use the three H-DNN architectures for inference as well as their training.

4.2.1 H-DNN Inference. The usage of H-DNNs during inference is the same as the original DNNs, except that human annotation is triggered and integrated into the model in case of a low confidence in inference. Participation in the inference stage is because human-annotated soft labels can help improve the detection accuracy of DNNs in challenging scenarios since humans can capture more detailed information. Also, note that we assume pool-based data processing. That is, the generated H-DNN will perform inference on batches of data, and human annotations are only necessary for a small portion of each batch during the inference. As shown in Figure 5a, if

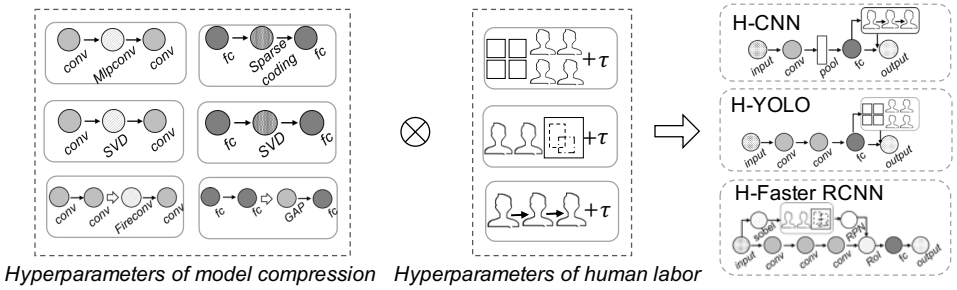


Fig. 6. H-Gen's search space design.

the confidence of the intermediate result is lower than a pre-defined threshold during inference, relevant data are sent for human annotation, which are used to correct and update the final output of the H-DNN.

Human annotation is implemented by three crowdsourcing task types: *majority voting* for H-CNN (see Section 4.1.1), *segmentation aggregation* for H-YOLO (see Section 4.1.2), and *region merging* for H-Faster-RCNN (see Section 4.1.3). We use WeSense [39] to manage the crowdsourcing tasks.

4.2.2 H-DNN Training. The training of H-DNNs differs from that of the original DNNs. It includes two stages. The first stage is the standard DNN training with the original training data. The second stage is the enhancement training stage. The human *does not directly* participate in the H-DNN's training stage. We utilize human annotations collected by human-in-the-loop inference as an extra knowledge base to guide the enhancement training of H-DNNs in a knowledge distillation manner, as in Figure 5b. In addition, we apply an AutoEncoder [45] to compare the feature distributions of the newly added human-annotated data with the dataset and filter the useless human-annotated samples in the enhancement training stage.

Specifically, for H-CNN and H-YOLO, since human annotations are stored in the form of results, we map the human-annotated data to a vector form corresponding to the model's output and train the model with the loss function of the high-temperature distillation process. For H-Faster-RCNN, since human annotations are stored in the form of attention area, we use the k-means++ clustering algorithm to complete the initialization of GMM on the human knowledge base and optimize the parameters through the EM [68] algorithm.

5 AUTOMATIC H-DNN OPTIMIZATION

This section presents H-Gen, an automatic hyperparameter optimization solution for H-DNNs. H-Gen is a NAS-style model compression framework that optimizes the performance (accuracy, latency) and the resource (storage, energy, and human labour cost) of H-DNNs (see Eq.(1)). As next, we discuss the search space (Section 5.1), metric calculation (Section 5.2), and search strategy (Section 5.3) in sequel.

5.1 Search Space

The search space of H-Gen consists of two categories of hyperparameters: model compression and human labor. As shown in Figure 6.

5.1.1 Hyperparameters of Compression Techniques. Model compression is a prevailing strategy to reduce the resource overhead of DNNs without notably compromising accuracy [12]. We follow prior studies [36] to search for the best compression technique combination for H-DNN generation because research [36, 38] showed that synergistic combination of multiple compression techniques may deliver better performance-resource trade-off. In H-Gen, we consider individual compression techniques as basic operations, and search for the best compression technique combination for each layer of a given H-DNN. For fair comparison, we integrate the same sets of

compression techniques as [36], which includes six compression techniques for convolutional (*conv*) layers and three for fully connected (fc) layers. H-Gen is a generic framework that can easily integrate other compression techniques. An exhausted inclusion of compression techniques is out of the scope of our work.

Specifically, the six compression techniques for *conv* layers are:

- o_c^1: decompose *conv* layers using convolution kernel sparse decomposition [33].
- o_c^2: replace the *conv* layers with depth-wise separable convolution [25].
- o_c^3: decompose *conv* layers using sparse random technique [9].
- o_c^4: replace *conv* with a Fire layer [26].
- o_c^5: replace *conv* with a Mlpconv [32].
- o_c^6: decompose the weight matrix of the *conv* using the SVD [29].

The three compression techniques for fc layers are:

- o_f^1: replace the fc layers with a global average pooling layer [32].
- o_f^2: decompose the weight matrix of the fc using the SVD [29].
- o_f^3: decompose the weight matrix of the fc using sparse-coding [6].

We define a discrete search space $\Phi = O_c^i \otimes O_f^j$ (see Eq.(1)) for the model compression techniques combination, where $\otimes$ means combination of compression techniques in different DNN layers. As an example, for a three-layer DNN ($conv_1$, $conv_2$, fc), there will be $6 \times 6 \times 3$ combinations of compression techniques.

5.1.2 Hyperparameters of Human Labour. One unique feature of H-Gen is the modeling of human labour as a new dimension of the model generation procedure. Recall that human annotation is triggered if the inference confidence (c_{H-CNN}, c_{H-YOLO} and $c_{H-Faster-RCNN}$ for H-CNN, H-YOLO, and H-Faster-RCNN, respectively) lower than a confidence threshold τ (see Section 4.1). Hence, it is natural to take the confidence threshold τ as the hyperparameter for human labour cost. This is reasonable because the lower confidence the inference is, the more human labour is involved in the model inference, which leads to longer average latency and higher monetary cost. Therefore, the confidence threshold determines the degree of human participation. In H-Gen, we normalize the value of threshold τ to a continuous real number space $[0, 1]$.

5.2 Metric Calculation

In this subsection, we explain how to calculate the various performance and resource metrics of H-DNNs, which include accuracy A, latency T, storage S, energy E, and human labour L. The hyperparameters of compression combination υ affect accuracy, latency, storage and energy. The human effort hyperparameter τ affects accuracy, latency and labour cost of the H-DNN. We assess human effort based on statistical methods and calculate the other metrics with the state-of-the-art models [36, 38].

5.2.1 Accuracy A. We directly measure the accuracy of a given H-CNN on a batch of test data. Similarly, we use the mean Average Precision (mAP) [49] in the test data as the accuracy for H-YOLO and H-Faster-RCNN.

5.2.2 Latency T. The latency of a given H-DNN consists of the time cost of DNN inference as well as that introduced by human participation (data transmission, crowdsourcing platform operation, human annotation, and feedback). We compute the average latency of all samples because H-DNN focuses on processing pool-based data. We calculate the latency of an H-DNN as follows:

$$T = T_{infer} + \frac{1}{\mathcal{M}} T_{extra} \tag{10}$$

where T_{infer} is the DNN inference time, which can be estimated as the sum of the delays of both the *conv* and *fc* layers [63]. The second term $\frac{1}{\mathcal{M}} T_{extra}$ is the average extra latency incurred by human participation in a test batch

with size $\mathcal{M}$. T_{extra} is the sum of extra latency incurred by human participation in a test batch, (determined by the samples to be annotated under the threshold τ and the time cost per sample).

5.2.3 Storage S. We apply the storage model in [38] to calculate the storage of a given H-DNN. Specifically, the storage S is derived by the total number of bits of activations and weights of all layers, as shown in Eq.(11):

$$S = \mathcal{S}_f + \mathcal{S}_p = B_a \times \mathcal{S}_a + B_w \times \mathcal{S}_w \tag{11}$$

where $\mathcal{S}_f$ and $\mathcal{S}_p$ are the storage for the activations and weights. B_a and B_w are the number of bits per activation and weight of the DNN during execution (*e.g.* 32 in Tensorflow). $\mathcal{S}_a$ and $\mathcal{S}_w$ denote the total number of activations and weights in all layers in the DNN.

5.2.4 Energy E. We combine the energy model in [38] and [63] to estimate the energy consumption of a given H-DNN. The energy cost E mainly comes the calculations in the DNN and the memory accesses, which can be estimated as follows.

$$E = \eta \cdot MACs + 200 \cdot \eta \cdot \mathcal{S}_f + 6 \cdot \eta \cdot \mathcal{S}_p \tag{12}$$

where $MACs$ is the number of multiply-accumulate (MAC) operations and η is the energy cost per MAC (52.8 PJ in mainstream mobile devices [38]). $\mathcal{S}_f$ and $\mathcal{S}_p$ are the storage for the activations and weights as in Eq.(11). Empirical studies [63] show that the energy costs to access activations and weights are roughly 200× and 6× of η, if activations are stored in DRAM while weights in cache.

5.2.5 Human Labour L. We calculate the labor cost as the ratio between the sum of the actual feedback costs and the maximum acceptable feedback cost in a batch, *i.e.*, the degree of human involvement:

$$L = \frac{L_{actual}}{L_{max}} \tag{13}$$

where L_{actual} is the labor cost actually consumed in a batch inference, determined by the actual number N of sample be annotated in a batch (affacted by the hyperparameter τ) and the human labor cost per sample (defined by Eq.(3), Eq.(6) and Eq.(9)). L_{max} is the maximum acceptable annotation cost in a batch, which is determined by the batch size, the maximum allowable number of participants per inference (default is 4), and the maximum number of operations per feedback (3 by default). Therefore, the human labour L for H-CNN, H-YOLO, and H-Faster-RCNN model is $L_{H-CNN} = \frac{N*L^0_{H-CNN}}{L_{max}}$, $L_{H-YOLO} = \frac{N*L^0_{H-YOLO}}{L_{max}}$, and $L_{H-Faster-RCNN} = \frac{N*L^0_{H-Faster-RCNN}}{L_{max}}$, respectively. Note that $L = 0$ means no human participation, and $L = 1$ implies that all $\mathcal{M}$ inputs in a batch require human involvement with maximum acceptable annotation cost per sample.

5.3 Search Strategy

Given the search space defined in Section 5.1, H-Gen adopts a *two-stage* strategy to search the hyperparameters for the optimization problem defined in Eq.(1). Specifically, H-Gen *alternately* selects the compression technology combination hyperparameters to build the DNN backbone architecture, and then determines the human labour hyperparameter based on the backbone architecture. The two-stage optimization decouples the intervened impact of two hyperparameter types and we apply deep reinforcement learning to effectively optimize the hyperparameters. Algorithm 1 illustrates the two-stage optimization workflow. It adopts the dueling DQN to select compression combination hyperparameters in the first stage. And it uses DDPG to select human labor hyperparameters in the second stage, as explained in detail below.

5.3.1 DQN for Compression Combination Hyperparameter. We adopt a deep Q-network (DQN) [43] to optimize the compression combination hyperparameter υ from a discrete search space. Inspired by [55], we utilize a DQN agent to compress the DNN in a layer-wise manner. The elements of the corresponding DQN in our problem context are defined as follows.

Algorithm 1: Two-stage Optimization for H-DNN generation

Input: $(A_d, T_d, S_c, E_c, L_c)$, H-DNN architecture, Dateset
Output: H-DNN with optimized configuration (architecture and amount of human labour) and trained weights

Initialize DQN, $DDPG$, H-DNN
for *episode = 1* $\rightarrow$ *1000* **do**
  Stage One:
  while *layer l is not the last layer in DNN* **do**
    $s_l \leftarrow DNN.layer_l.info$
    select an action $a_l = argmax_a Q(s_l, a_l; \omega)$
    compress the layer by a_t
    l++;
  end
  calculate the metrics E, S with compressed DNN architecture
  Stage Two:
  train the compressed H-DNN on dataset and human knowledge base
  $s_i' \leftarrow DNN.compression.combination$
  select a_i from O' at s_i by $actor$ with noise
  forward the H-DNN
  calculate the metrics A, T, E, S, L to compute Reward R_{1i}, R_{2i}.
  broadcast R_{1i}, R_i to be the reward all of s_l and s_i'
  sample random minibatch of transitions from experience memory
  update DQN and $DDPG$ via learning step
  every num reset target network of DQN and $DDPG$
  episode++
  if *A, T, E, S, L satisfy user demand* **then**
    break
  end
end

- *State:* We define state $s_l = \{l, h_l, w_l, c_l\}$, where l is the layer index, and h_l, w_t, c_l represent the height, width, and channels of input features to layer l, respectively. For one-dimensional input features to fc layer, $w_l = 1$ and $c_l = 1$.
- *Action:* For each layer l, action $o_l \in O$ is defined as the optional compression technology in layer l, where O is the set of compression techniques in Section 5.1.1.
- *Reward:* Following [38], we convert the original constrained optimization problem in Eq.(1) into an unconstrained one as a dueling DQN. Maximizing Eq.(1) by DQN can cause ambiguity (as mentioned in AdaDeep), so we adopt the dueling DQN to separate the state-action value function and the state-action advantage function into two parallel streams to better separate constraints and goals. We define the final reward as the sum of the following two parts (*i.e.*, objective gain R_1 and constraint satisfaction R_2):

$$R_1 = \delta_1 log(A - A_d) + \delta_2 log(T_d - T) \tag{14}$$

$$R_2 = \delta_3 log(E_c - E) + \delta_4 log(S_c - S) + \delta_5 log(L_c - L) \tag{15}$$

  where, δ_3, δ_4, and δ_5 are the coefficients for the weighted summation of three constraints.

The agent receives a state s_l for each layer l and then outputs an optimal action o_l, *i.e.*, the selected compression technique in layer l. Then it searches for the next layer, receiving new states s_{l+1} until all layers have been explored. Afterwards, we train the DNN on the input dataset, calculate its metrics as the global reward R_{1i} and R_{2i}, and finally return it to the agent. As a separate note, once the constraints R_2 are satisfied, the optimal performance object R_1 is the most critical goal in picking a sole solution during the search process.

The parameter weights ω of dueling DQN [57] is trained by iteratively minimizing the loss function as follows:

$$loss_{DQN} = \mathbb{E}_{s_i.a_i,r,s_{i+1}} \left[(\bar{Q}_{DQN} - Q(s, a; \omega))^2 \right] \tag{16}$$

where $\bar{Q}_{DQN} = R_{1i} + R_{2i} + \gamma \max_{a_{i+1}} Q(s_{i+1}, a_{i+1}; \omega); \omega^{-})$ is the target for the current iteration i. Note that ω of dueling DQN represent share $conv$ layers and two streams of fc layers. γ is decay factor for future rewards (0.01 by default). We use stochastic gradient descent to optimize this function. During the optimization process, we choose a maximum Q-value action by $\varepsilon - greedy$ policy and a random action by probability $1 - \varepsilon$ and store the agent's experiences in memory.

5.3.2 DDPG for Human Labor Hyperparameters. After training the weights of the compressed DNN, we explore the human participation threshold τ for the optimal performance of H-DNN in the second stage. Since deep deterministic policy gradient (DDPG) performs well in solving continuous control problems and can be jointly trained end-to-end with DQN, we adopt it to optimize the human effort hyperparameter. It follows an actor-critic framework to concurrently learn the actor network and the value-based critic network. The actor network gets advice from the critic network that helps the actor network decide which actions to reinforce during training. Meanwhile, the DDPG makes uses of actor networks and critic networks to improve the stability and efficiency of training. The relevant elements are defined as follows.

- *State:* The compression combination hyperparameter of the compression technique selected by DQN.
- *Action:* $O' \sim [0, 1]$ is a space of real numbers representing the range of human participation threshold.
- *Reward:* It is same as DQN as defined in Eq.(14).

DDPG observes a compression combination hyperparameter state s'_i, and leverages the DDPG's predict actor network to estimate the deterministic optimal action o'_i with truncated normal distribution noise [22]. To train such a DDPG optimizer with the parameters ω^{actor} and ω^{critic}, we optimize the actor network at iteration i by the policy gradient [51]. And we train the critic network by optimizing the loss function $loss$ from both the random replay memory and the output of the actor and the critic networks:

$$y_i = R_{1i} + R_{2i} + \gamma \bar{Q}(s_{i+1}, \bar{A}_{DDPG}(s'_{i+1}; \bar{\omega}^{actor})) \tag{17}$$

$$loss_{DDPG} = \frac{1}{N_{batch}} \sum_i (y_i - Q_{DDPG}(s'_i, o'_i; \omega^{critic}))^2 \tag{18}$$

where N_{batch} is the minibatch size of transitions. y_i is computed by the sum of immediate reward R_{1i} and R_{2i} and the outputs of the frozen actor and critic. We initialize the replay memory with 10000 historical transitions. s is the compression combination hyperparameters selected by dueling DQN. a is the selective action in state s, which is a space of real numbers representing the range of the human participation threshold τ. r is the reward, the sum of the objective gain R_1 and constraint satisfaction R_2. s' is the compression combination hyperparameters selected by the dueling DQN in the next iteration. During the search process, we sample a random mini-batch (default as 32) of transitions from the replay memory to update the agent parameters, breaking the correlation of the transition sequence and achieving faster convergence.

6 IMPLEMENTATION

We implement the H-Gen's framework for H-DNN model generation with Tensorflow and Keras in Python. The implementation of H-Gen consists of *server side* and *mobile application side*, as shown in Figure 7.

- The server side realizes the automatic generation of optimized H-DNN and the model's training. The server connected to mobile applications and crowdsourcing platform by Socket, and maintain a human knowledge base by NoSQL. The server side is hosted on a server with the Intel i9-10900K 3.70GHz CPU and NVIDIA GTX 3090Ti GPU.
- For the mobile application side, during the inference phase of the generated H-DNNs, we implement the human-in-the-loop inference with TensorFlow Lite in JAVA as Android apps on the mobile platforms. They exchange intermediate data using JAVA Socket. Human participation for annotation (*i.e.*, crowdsourcing)

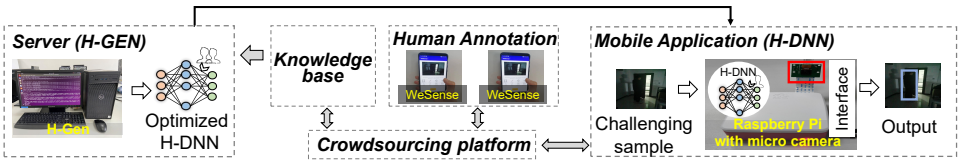


Fig. 7. The implementation of H-Gen.

Table 1. Dataset overview.

NO.	Type	(Challenging) tasks	Datasets	Description (Sample number in training/test/class number)
D1	Classification	Digit	MNIST	60,000 images with the shape of 28×28 (55,000/10,000/10)
D2		Noisy digit	Noisy MNIST	60,000 images processed with Gaussian noise (55,000/10,000/10)
D3		Cloth	FashionMNIST	60,000 images with the shape of 28×28 (55,000/10,000/10)
D4		Noisy cloth	Noisy FashionMNIST	60,000 images processed with Gaussian noise (55,000/10,000/10)
D5		Low-resolution cloth	Low-resolution FashionMNIST	60,000 images re-scaled by bilinear sampling (55,000/10,000/10)
D6	Object detection	Small object	VisDrone-DET2019	2,000 images randomly selected (1,600/400/10)
D7		Low-light object	ExDark	2,000 low-light images randomly selected (1,600/400/12)
D8		Camouflaged object	COD-10k	2,000 images randomly selected (1,600/400/20)
D9		Motion-blur object	ImageNet VID	2,000 images randomly selected (1,600/400/20)
D10		Infrared object	FLIR Thermal dataset	10,288 infrared images (8,862/1,366/15)
D11		Low-resolution object	Low-resolution VOC dataset	9,953 images processed with Gaussian noise (5,001/4,952/20)
D12		Noisy object	Noisy VOC dataset	9,953 images re-scaled by bilinear sampling (5,001/4,952/20)

is implemented and managed by the Amazon MTurk, which supports task generation and participant management. And the human-machine interaction is implemented in a self-developed mobile APP, WeSense.

H-Gen works as follows. When H-Gen performs H-DNN generation, it iteratively selects hyperparameters (*i.e.*, actions) and controls the training and inference of H-DNNs. For the training of H-DNNs, the training dataset and human knowledge base (initialized with 5% of the public dataset) are used in each iteration because their labels are in the same form. During inference, human participants will be involved. The data annotated in the inference phase are sent to the crowdsourcing platform to publish crowdsourcing tasks. Human participants accept crowdsourcing tasks through a self-developed APP WeSense [39] to complete the annotation and feedback to the server-side. These annotations help the model in inference and accuracy calculations when updating the human knowledge base. When new human knowledge accumulates to a preset threshold(*e.g.* 200 for H-CNN, 1000 for H-YOLO, and 1000 for H-Faster RCNN), H-Gen will retrain H-DNNs. H-Gen will use new labeled data to fine-tune H-DNN weights and update the new H-DNN to mobile devices. We assign different incentive levels (*e.g.* 2,4,6) for each crowd response according to various annotation tasks. We count the extra latency from intermediate data generation and uploading for task release and annotation feedback to the completion of inference. Then we calculate the average inference latency based on the number of samples and the statistics of the extra latency introduced by human feedback.

7 EVALUATION

7.1 Experimental Setups

7.1.1 Challenging Tasks and Datasets. Table 1 summarizes the datasets used in our evaluation. As mentioned in Section 4, we showcase H-Gen with two machine learning tasks, *i.e.*, classification, and object detection on the following twelve challenging (*i.e.*, $D2, D4, D5, and D6 \sim D12$) and standard (*i.e.*, $D1, D3$) tasks.

- For classification tasks and the corresponding H-CNN architecture, we adopt five datasets: the public digit recognition dataset MNIST (D1), the MNIST with artificially added Gaussian noises with variance 0.2 (D2), the public cloth classification dataset FashionMNIST (D3), FashionMNIST added with artificially added Gaussian noises variance 0.2 (D4), and FashionMNIST with low resolution to a quarter re-scaled by

Table 2. Performance comparison between AdaDeep and H-Gen for different model architectures.

Mobile tasks	Original DNNs	Frameworks	Performance of the generated H-DNN				
			A(%)	T(ms)	E(mJ)	S(MB)	Labor Cost(%)
Classification	CNN (LeNet)	Origin (Baseline)	82.1	22.3	2.4	26.1	——
		AdaDeep(Baseline)	81.5	17.8	1.7	1.6	——
		H-Gen	89.4	28.3	1.5	1.4	2.1
Object Detection	YOLO (v1)	Origin (Baseline)	35.1	702.1	330.7	1640.8	——
		AdaDeep (Baseline)	34.2	320.4	260.1	253.7	——
		H-Gen	38.6	375.8	197.3	231.6	2.9
	Faster RCNN (VGG backbone)	Origin (Baseline)	39.2	836.3	420.6	1730.5	——
		AdaDeep (Baseline)	37.8	504.7	321.5	442.2	——
		H-Gen	42.4	497.3	301.2	351.7	1.7

bilinear sampling (D5). Here, we artificially add noises to simulate the impact of sensor noise caused by poor lighting or high temperature in real-world mobile applications.

- For object detection tasks and the corresponding H-DNN architectures, *i.e.*, H-YOLO and H-Faster-RCNN, we employ seven datasets, which represent several challenging object recognition scenarios, including small object detection (D6: Visdrone), low-light object detection (D7: Exclusively Dark), camouflaged object detection (D8:COD-10k), motion blur object detection (D9: ImageNet VID), thermal object detection (D10:FLIR Thermal), low-resolution object detection (D11: low-resolution VOC), and noisy object detection (D12: noisy VOC). We downscale the original images to a quarter pixel and use bilinear sampling to rescale them to the original size for generating the low-resolution vision of VOC datasets [17]. We add the Gaussian noise with a variance of 0.2 to VOC images to generate the noisy VOC (D12).

We use LeNet [30], YOLOv1 (VGG16 backbone network) [48], and Faster RCNN (VGG16 backbone network) [49], as the origin architectures for H-CNN, H-YOLO, and H-Faster-RCNN, respectively, to input into H-Gen for automatic model generation and optimization.

7.1.2 Mobile and Embedded Platforms. We experiment with three mobile and embedded platforms, *i.e.*, MingDong smartwatch (Device 1), Huawei P20 (Device 2), and Raspberry Pi 4B (Device 3). They have various resource constraints. Specifically, the MingDong X361 is equipped with Cortex-A7 processor, 3G DRAM, 1MB L2-Cache and 1000mA battery. The Huawei P20 has a Kirin 970 processor, 6G DRAM, 2MB L2-Cache, and 3400mA battery, and the Raspberry Pi 4B has the Cortex-A72 processor, 8G DRAM, 2MB L2-Cache, and 3800mA battery.

7.1.3 Comparison Baseline. We adopt three original DNNs (see § 7.1.1) and nine compression techniques (see § 5.1.1) as baselines for performance (accuracy, latency, energy, storage) comparison. In addition, we compare H-Gen with AdaDeep [36, 38], an automated DNN generation and compression framework with diverse user-specified mobile application performance requirements (*e.g.* accuracy, latency) and mobile platform-imposed resource constraints (*e.g.* computation, storage, and energy budgets). By comparing with it, we show how H-Gen break the accuracy restriction by introducing human involvement, which improves the accuracy of the uncompressed model in case of challenging input samples.

7.2 Overall Performance Comparison with Baselines

7.2.1 Performance Comparison for Different Model Architectures. We compare the performance of different models generated by AdaDeep and H-Gen (L_c is set to 3.0%) for a specific mobile device (Device 2). We selected three original DNNs and three AdaDeep-optimized DNNs as the baseline for the experiments in this section. For classification tasks, we use LeNet on D5. For object detection tasks, we use YOLOv1 and Faster RCNN (VGG16 backbone) on D6. We choose the same *conv* and fc layers as their compression space for a fair comparison. Specifically, we select the compression technique of CNN layer by layer. For YOLOv1 and Faster RCNN, whose

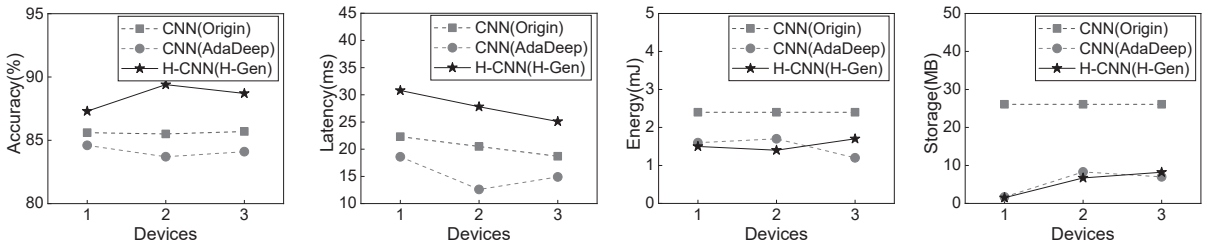
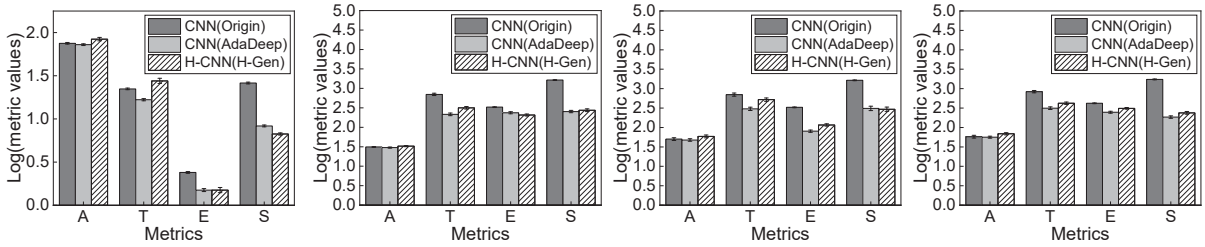


Fig. 8. Performance comparison over different mobile devices.

backbone consists of 5 blocks (block1-5), we skip the first *conv* layer of each block to preserve more feature information. Table 2 shows the experimental results. First, compare to origin DNNs, for different tasks and models, H-DNNs generated by H-Gen improve accuracy by 3.2% $\sim$ 7.3%, and reduce storage by 24.7 $\sim$ 1378.8MB, latency by $-6.0 \sim 381.7ms$, energy cost by 0.9 $\sim$ 133.4mJ with labor cost 1.7% $\sim$ 2.1%. Second, compare to AdaDeep, H-DNNs generated by H-Gen improve accuracy by 4.4% $\sim$ 7.9%, and reduce storage by 0.2 $\sim$ 90.5MB and energy cost by 0.2 $\sim$ 62.8mJ with extra latency $-7.4 \sim 55.4ms$. H-Gen takes 1.5 hours to generate H-CNN, 6 hours to generate H-YOLO, and 8 hours to generate H-Faster-RCNN, respectively, on a server with the Intel i9-10900K 3.70GHz CPU and NVIDIA GTX 3090Ti GPU. In summery, H-Gen achieves the best overall trade-off between inference accuracy, energy cost, and storage for different models with little labor cost and extra latency. This is because the involved human intelligence in H-Gen pushes the DNN's accuracy-resource trade-off limit.

7.2.2 Performance Comparison over Different Mobile Devices. We compare the performance of AdaDeep and H-Gen to generate a specific deep model (CNN) for different mobile devices (Device 1-3). In this thread of experiments, we perform H-CNN generation on low-resolution classification task (D5) based on LeNet. And the labor budget L_c is set as 3.0%. Figure 8 shows the results. First, compared with AdaDeep, which loses 0.9% $\sim$ 1.9% accuracy on different devices, H-Gen can improve the accuracy of the original DNN by 1.8% $\sim$ 3.3% with human labor. Second, compared to the original DNN, H-Gen can reduce 0.7 $\sim$ 1.0mJ energy consumption and 17.9 $\sim$ 24.4MB storage and achieve similar effects on energy and storage to AdaDeep on different devices. It is worth mentioning that H-Gen achieves a better resource trade-off on most devices (Device 1-2). Third, due to the human labor in DNN inference, H-CNN generated by H-Gen will introduce 1.5 $\sim$ 8.5ms extra latency, which is acceptable. The batch size in this experiment is 10,000, where 21 samples are sent for human annotations, and the average latency is 29.4ms, in which the extra human annotation latency is 8.5ms. In summary, H-Gen can achieve *higher accuracy* under different device resource constraints, compared with the baselines.

7.2.3 Performance Comparison on Practical Challenging Tasks. We compare the performance of AdaDeep and H-Gen to generate DNNs for different challenging tasks (on Device 3). For classification tasks, we conduct experiments on D2 (H-CNN), and for object detection tasks, we conduct experiments on D7 (H-YOLO), D9 (H-YOLO), and D12 (H-Faster RCNN). Figure 9 shows the experimental results. First, compared with AdaDeep, which loses 1.2% $\sim$ 2.9% accuracy on all tasks, H-Gen can improve the accuracy by 2.7% $\sim$ 5.1% with human labor. Second, for different tasks, H-Gen achieves similar energy and parameter size reductions to AdaDeep. Third, for the object detection tasks, although the latency of H-DNN is slightly higher than the model generated by AdaDeep, it is still significantly reduced for the original DNN. In summary, H-Gen yields the highest accuracy for different tasks, especially on challenging input samples, while satisfying the device resource constraints.



(a) D2 (b) D7 (c) D9 (d) D10

Fig. 9. Performance comparison on practical challenging tasks.

7.2.4 Performance Comparison on Compression Performance. We experiment to illustrate the impact of human-annotated soft labels on compression technique choice and compression performance (*i.e.*, accuracy A, latency T, energy E, and storage S). Specifically, we compare the model performance generated by AdaDeep (without human-annotated soft labels) and H-Gen (with human-annotated soft labels) to satisfy the $\geq$ 90% accuracy demands on task D4. Both of them satisfy the accuracy requirement. However, H-Gen reduces (improves) the generated H-CNN's energy cost by $0.9mJ$ (39%), and storage by $7.3MB$ (80%), compared with AdaDeep. This shows that the accuracy gain brought by human annotation in H-Gen can balance the accuracy loss caused by model compression, thereby improving the overall compression performance.

Table 3. Performance comparison of CNN generated by AdaDeep and H-Gen with $\geq$ 90% accuracy demands.

Framework	Compression combination	A(%)	T(ms)	E(mJ)	S(MB)
AdaDeep (without human participation)	$o_1^c + o_1^f$	90.1%	21.3	2.3	9.1
H-Gen (with human participation)	$o_4^c + o_2^f$	90.4%	35.3	1.4	1.8

7.3 A Closer Look at H-DNNs Generated by H-Gen

7.3.1 H-DNNs with Different Performance Requirements. This experiment evaluates H-Gen with different object detection challenging tasks (and thus diverse performance requirements) under a fixed human labor budget 6% for Device 3, using H-YOLO and H-Faster-RCNN. We define $\delta_3 \sim \delta_5 = 0.3, 0.3, 0.4$ of Eq.(14). Table 4 shows the details and the performance of H-YOLO and H-Faster-RCNN generated by H-Gen. First, compared with the origin YOLO, H-YOLO generated by H-Gen in different tasks can improve accuracy by 0.6% $\sim$ 5.3% and reduce parameter size by 4.3× $\sim$ 12.4×, latency by 1.3× $\sim$ 2.9×, energy cost by 1.1× $\sim$ 2.8× with labor cost 0.8% $\sim$ 4.1%. Second, compared with the origin Faster RCNN, the H-Faster-RCNN generated by H-Gen can improve accuracy by 1.1% $\sim$ 4.7% and reduce parameter size by 3.2× $\sim$ 10.8×, energy cost by 1.1× $\sim$ 1.7× with labor cost 0.6% $\sim$ 4.4%. In conclusion, H-Gen can improve accuracy and reduce latency by tuning H-DNN model hyperparameters in various tasks.

7.3.2 H-DNNs with Different Platform Resource Constraints. This experiment evaluates H-Gen with different mobile devices (and thus platform-imposed resource constraints) under a fixed human labor budget $L_c = 3\%$, using H-CNN and FashionMNIST (D3) as the model and dataset. Different mobile devices affect $\delta_3 \sim \delta_5$ of Eq.(14). Under fixed human labour budget $L_c = 3\%$, $\delta_5 = \frac{0.1 - L_c}{0.1}$. $\delta_4 = max\{\frac{3800 - E_{battery}}{3800}, 0.5\} * (1 - \delta_5)$, and $\delta_3 = 1 - \delta_4 - \delta_5$. Table 5 shows the performance of H-CNN generated by H-Gen. First, compared with the origin CNN ($A = 90.4\%, T = 22.3ms, E = 2.4mJ, S = 26.1MB$), H-CNN generated by H-Gen can improve accuracy by 1.6% $\sim$ 4.3% and reduce parameter size by 1.9× $\sim$ 13.7×, energy cost by 1.3× $\sim$ 1.6× with labor cost 1.6% $\sim$ 2.5%. Second,

Table 4. Performance of H-DNNs generated by H-Gen in different object detection datasets. (↑) represents accuracy improvement and (×) represents the cost reduction.

Task	H-YOLO(Compared with origin YOLO)					H-Faster RCNN(Compared with origin Faster RCNN)				
	A(%)	T(ms)	E(mJ)	S(MB)	L(%)	A(%)	T(ms)	E(mJ)	S(MB)	L(%)
D6	39.2(↑ 4.2)	545.4(1.5×)	197.8(1.7×)	201.5(8.1×)	3.4	42.1(↑ 3.1)	761.5(1.1×)	321.2(1.3×)	442.7(3.9×)	3.8
D7	32.8(↑ 1.5)	316.7(2.6×)	206.7(1.6×)	274.9(6.0×)	1.6	36.8(↑ 2.4)	425.4(1.9×)	286.6(1.5×)	363.2(4.8×)	2.4
D8	28.7(↑ 0.6)	283.7(2.9×)	306.8(1.1×)	382.4(4.3×)	0.8	30.2(↑ 1.1)	364.9(2.3×)	367.1(1.1×)	534.4(3.2×)	0.6
D9	52.4(↑ 2.1)	521.3(1.6×)	116.1(2.8×)	294.7(5.6×)	2.1	58.8(↑ 4.2)	624.2(1.3×)	241.4(1.7×)	160.1(10.8×)	4.4
D10	51.5(↑ 5.3)	668.5(1.3×)	285.4(1.2×)	132.2(12.4×)	4.1	56.4(↑ 2.5)	357.3(2.3×)	301.9(1.4×)	221.6(7.8×)	1.8
D11	50.1(↑ 3.0)	486.7(1.7×)	252.7(1.3×)	227.4(7.2×)	2.8	60.7(↑ 4.7)	526.7(1.6×)	354.2(1.2×)	180.7(9.6×)	2.8
D12	56.7(↑ 2.8)	511.9(1.6×)	163.2(2.0×)	189.9(8.6×)	3.7	61.5(↑ 3.6)	421.1(2.0×)	310.8(1.4×)	238.2(7.3×)	1.7

Table 5. Performance of H-DNNs generated by H-Gen with different platform-imposed resource budgets.

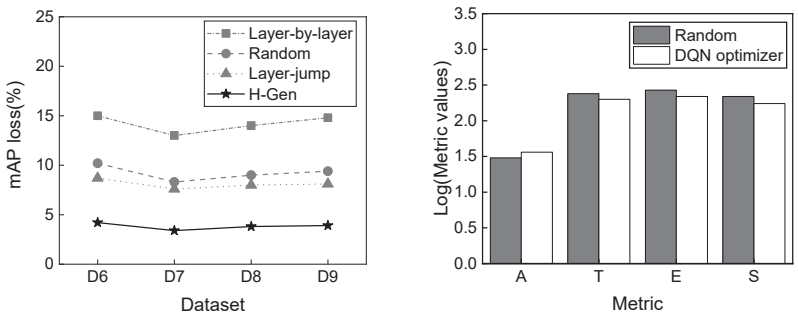
Mobile Devices	H-CNN hyperparameters		Compared with the origin CNN				
	Compression combination	human labor τ	A	T	E	S	L
Device 1	$o_4^c + o_1^f$	0.38	↑ 1.60%	0.8×	1.6×	13.7×	2.5%
Device 2	$o_5^c + o_2^f$	0.22	↑ 3.20%	1.2×	1.3×	1.9×	1.2%
Device 3	$o_2^c + o_1^f$	0.26	↑ 4.30%	1.1×	1.4×	2.3×	1.6%

Table 6. Performance of H-DNNs generated by H-Gen with different human labour budgets.

Labor budget L_c(%)	H-YOLO hyperparameters		Compared with the origin YOLO				
	Compression Combination	Human labor τ	A	T	E	S	L
0	$o_5^c + o_1^f$	-	↓ 0.90%	2.2×	1.3×	5.6×	0
2	$o_6^c + o_5^c + o_2^f$	0.23	↑ 1.30%	2.4×	1.4×	7.1×	1.60%
4	$o_2^c + o_1^f$	0.48	↑ 4.20%	1.9×	1.9×	6.1×	2.40%
6	$o_1^c + o_5^c + o_3^f$	0.55	↑ 4.70%	2.0×	1.7×	8.6×	4.20%
8	$o_5^c + o_2^c + o_1^f$	0.61	↑ 4.70%	1.4×	2.2×	10.8×	6.80%

for Device 1 with fewer resources, the generated model achieves up to a 1.6× reduction in energy consumption and 13.7× reduction in parameter size with a 1.6% improvement accuracy. Third, H-Gen automatically searches for suitable H-CNN hyperparameters, *e.g.* o_4^c to *conv*, o_1^f to fc and 0.38 to τ for Device 1, o_5^c to *conv*, o_2^f to fc and 0.22 to τ for Device 2. In summary, H-Gen can maximize accuracy and reduce overall latency by adaptively generating H-DNNs that meet their resource constraints for different mobile devices.

7.3.3 H-DNNs with Different Human Labor Budgets. This experiment evaluates H-Gen with different labor budget with Device 2, using H-YOLO and VisDrone (D6) as the model and dataset. Different labor budget L_c affects $\delta_3 \sim \delta_5$ of Eq.(14). Specifically, $\delta_5 = \frac{0.1-L_c}{0.1}$, $\delta_3 + \delta_4 = 1 - \frac{0.1-L_c}{0.1}$ and the ratio of δ_3 and δ_4 is determined by the resource of Device 2. Table 6 shows the details and performance of H-YOLO generated by H-Gen. Compared with the origin YOLO, H-YOLO generated by H-Gen can improve mAP by 1.3% $\sim$ 4.9% and reduce parameter size by 6.1× $\sim$ 10.8×, latency by 1.4× $\sim$ 2.4×, energy cost by 1.4× $\sim$ 2.2× with labor cost 1.6% $\sim$ 6.8%. Also, H-Gen automatically searches for suitable H-DNN hyperparameters, *e.g.* o_6^c to *conv* of H-YOLO network $block1-2$, o_5^c to *conv* of H-YOLO network $block3-5$, o_2^f to H-YOLO network fc and 0.23 to τ for $L_c = 2\%$; o_5^c to *conv* of H-YOLO network $block1-2$, o_2^c to *conv* of H-YOLO network $block3-5$, o_1^f to H-YOLO network fc and 0.61 to τ for $L_c = 8\%$. In conclusion, H-Gen can maximize accuracy and reduce latency by automatically selecting model hyperparameters that meet their different labor budget for mobile applications.



(a) layer compression selection (b) Combination optimizer

Fig. 10. Impact of compression hyperparameter: (a) layer compression selection and (b) compression combination optimizer.

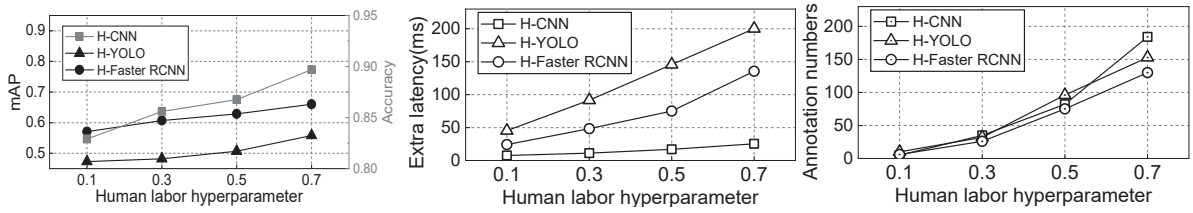
7.4 Parameter Sensitivity Analysis of H-Gen

7.4.1 Impact of Compression Layer Selection. For classification tasks, AdaDeep [38] has shown that layer-wise selective compression techniques only incur a slight accuracy loss. However, more feature information needs to be preserved for more complex object detection tasks. Therefore, these experiments compare the impact of different layer compression selection methods on the accuracy. We use the o_5^c compression technology to compress the backbone *conv* layer of Faster RCNN. We compare four methods of layer compression selection, *i.e.*, layer-by-layer (each layer is compressed), random (each layer is compressed with a probability of 0.5), layer jump (compress one layer every two layers), and H-Gen (skip the first layer of each block). The results are shown in Figure 10a. While other methods incur unacceptable mAP losses, H-Gen's compression layer selection can achieve minimum mAP loss 3.4% $\sim$ 4.2% under the condition that the optional compression layers are similar.

7.4.2 Impact of Compression Combination Optimizer. Adadeep [38] has verified the advantages of compression technology combination through experiments for CNN. This experiment is to verify that the compression combination is also effective for more complex object detection models in H-Gen. We compare the performance of the random combination of compression methods for the YOLO model on D6 with compression methods selected by the optimizer. The results are shown in Figure 10b. Compared to a random combination of compression methods, the optimized combination improves 3.4% mAP and reduces 40.5MB parameter size, 30.7ms latency, and 31.8mJ energy cost. In conclusion, the optimized combination can satisfy different resource constraints.

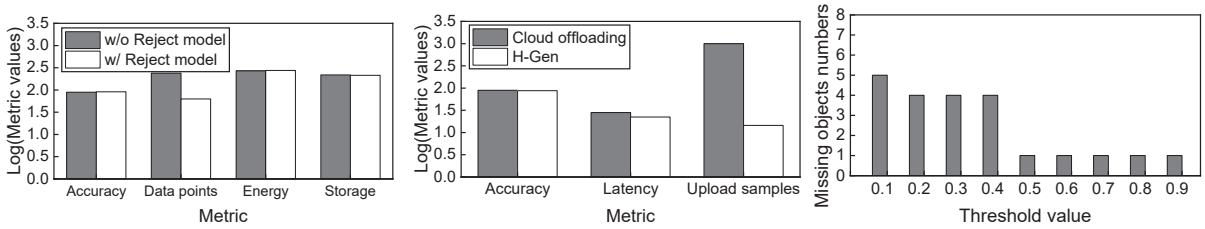
7.4.3 Impact of Human Labor Hyperparameter. This experiment aims to illustrate the impact of human labor hyperparameters on accuracy, extra latency, and the number of annotations of H-DNNs. For different H-DNNs, we calculate the confidence (Eq.(2), Eq.(4) and Eq.(7)) of each inference, and take confidence threshold τ as the hyperparameter for human labor. Figure 11 shows the results. As mentioned in Section 5.2, the accuracy, extra latency, and annotation numbers are related to different human labor hyperparameter τ. Both of them increase with the increase of τ. Taking H-CNN as an example, we conducted 10,000 inferences. When the τ is set as 0.1, the number of human annotations required is 11. Accordingly, the model's accuracy is 82.9%, and the extra latency caused by human annotations is 7.5ms. And when the τ is 0.5, the number of required human annotations increased to 76. The H-CNN's accuracy rose to 84.1%, and the extra latency is about 26.9ms. The experiment results show that the human labor hyperparameter τ is tunable for optimizing H-DNN's accuracy and latency and proves that we use it to trade-off the performance.

7.4.4 Impact of the Filter Model in the Enhancement Training Stage. We verify the impact of introducing the filter model in the human knowledge base through an ablation study with H-CNN on D2. Figure 12a shows the



(a) accuracy (b) latency (c) annotation numbers

Fig. 11. Impact of human labor hyperparameter on (a) accuracy, (b) latency and (c) annotation numbers.



(a) Impact of the filter model (b) Comparison with cloud (c) Impact of the threshold value

Fig. 12. Ablation studies.

results. Introducing the filter model reduces the number of selected human-annotated samples for training (*e.g.* from 52 to 36), which improves the training speed by 28.4%. Also, it has little impact on the generated H-DNN's accuracy and compression performance. In summary, the filter model can speed up the training speed.

7.4.5 Comparison with Cloud-based Method. We compare the crowdsourcing-based human-in-the-loop inference scheme with the cloud-assisted baseline. In the cloud-based scheme, we transmit the data (1000 images) from the smartphone (Device 2) to the cloud server for inference. Figure 12b shows the comparison results. In the H-YOLO's inference scheme, we only need to upload a small number of samples (4.6% in a batch size of 1000) with low inference confidence to the crowdsourcing platform for human annotations. The average latency is about 184.4*ms*. Compared to the cloud-assisted method, h-YOLO dramatically saves the transmission cost with competitive accuracy ($\leq$ 1.3% accuracy drop) and latency ($\leq$ 6*ms* less latency).

7.4.6 Impact of Threshold Value in Confidence Calculation. We calculate the input's overall confidence in H-YOLO by accumulating each grid's classification and regression confidence, as mentioned in Section 4.1.3. This experiment explores the number of missing objects in a grid under different thresholds. We counted the number of H-YOLO missing objects under different threshold settings within a batch (*i.e.*, 1000) of samples. Figure 12c shows the experimental results. When the threshold is greater than 0.5, the average number of missing objects per input is $\leq$ 1. When the threshold is less than 0.5, the number of missing objects per input is greater than 4. Therefore, 0.5 is the maximum threshold that guarantees a low object missing rate. Accordingly, we set the threshold as 0.5 by default in the confidence calculation.

7.5 Necessity of Human Participation in H-DNNs

7.5.1 Performance of H-CNN. We use LeNet to evaluate the accuracy of the H-CNN in Section 4.1.1 on five tasks (D1~D5), *i.e.*, Digit, Fashion, and challenging factors (*e.g.* noise, low light) classification. The challenging factors

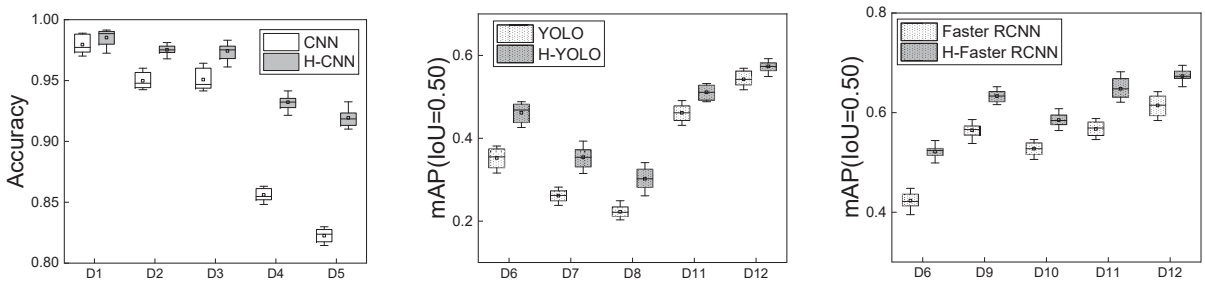


Fig. 13. Performance comparison of DNNs and the corresponding H-DNNs on challenging datasets.

in CNN will affect the misalignment of features, resulting in low accuracy. In H-CNN, we choose the same network hyperparameters (Optimized by Hyperopt) as the original LeNet, and fix the human labor hyperparameter τ to 0.2. As shown in Figure 13(a), we compare the accuracy of the origin CNN and H-CNN. First, for different classification tasks and challenging factors, human involvement can improve the accuracy of the original model by 0.3% $\sim$ 9.5%. Second, H-CNN can improve by 0.4% $\sim$ 6.4% on regular classification tasks (D1 and D3) and by 2.9% $\sim$ 9.5% on noisy and low-resolution tasks (D2, D4, and D5). Summaryly, human involvement can help CNN improve accuracy and play a more significant role under more challenging factors.

7.5.2 Performance of H-YOLO. We evaluate the mAP (IOU=0.5) of the H-YOLO in Section 4.1.2 on five different challenging tasks (D6, D7, D8, D11, and D12), *i.e.*, small, low-light, camouflaged, low resolution and noise object detection. These factors further aggravate the YOLO's shortcomings of low-accuracy detection for small objects. We fix the human labor hyperparameter τ to 0.3. As shown in Figure 13(b), we compare the performance of the origin YOLO and H-YOLO. First, for different challenging object detection tasks, human involvement can improve the mAP of the original DNN by 5.2% $\sim$ 11.1%. Second, H-YOLO can improve 11.1% on small object detection tasks (D6) and improve by 5.1% $\sim$ 9.4% on other challenging tasks (D7, D8, D11, and D12). In summary, human involvement can help the YOLO model improve the ability for different challenging objects and improve for small and camouflaged objects.

7.5.3 Performance of H-Faster-RCNN. We evaluate the mAP (IOU=0.5) of the H-Faster-RCNN in Section 4.1.3 on five different challenging tasks (D6, D9, D10, D11, and D12), *i.e.*, small, thermal, motion-blur, low resolution and noise object detection. We fix the human labor hyperparameter τ to 0.3. As shown in Figure 13(c), we compare the performance of the origin Faster RCNN and H-Faster-RCNN. For different challenging object detection tasks, human involvement can improve the mAP of the initial model by 4.3% $\sim$ 8.2%. In conclusion, human-annotated soft labels can help capture detailed information in challenging scenarios to improve accuracy.

7.6 Case Study

We deploy H-Gen on a server with the Intel i9-10900K 3.70GHz CPU and NVIDIA GTX 3090Ti GPU to generate the H-CNNs for a mobile emotion recognition application. We load the generated H-CNNs with an Android application on a smartphone (*i.e.*, Huawei P20, device2). The application employs a *pool-based data processing* method. Specifically, it adopts the generated H-CNN to recognize the user's expressions (*e.g.* anger, disgust, fear, happiness, routine, sadness, and surprise) per frame and infers the user's overall emotion based on the expression prediction results of 120 frames within a time window of 2s. Figure 14 illustrates the scenario.

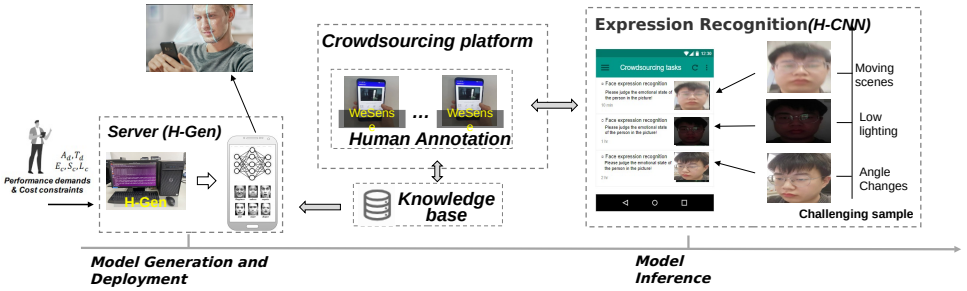


Fig. 14. Case study of H-Gen in the mobile user emotion detection application.

Table 7. H-CNN's performance over three challenging scenarios, in terms of (a) accuracy, (b)latency, (c) storage, (d) energy cost, and (e) # of annotations (in 2s).

Scenarios	A(%)	T(ms)	E(mJ)	S(MB)	# Annotations(in 2s)
Angle-changing	70.4	494.5	1.8	2.6	10
Moving scenes	70.1	498.5	1.3	1.4	14
Low lighting	70.9	488.5	1.9	2.8	8

In the training stage, H-Gen automatically optimizes the hyperparameters of H-CNN according to the user-specified performance demands and resource/labor budgets (*i.e.*, A_d=70%, $T_d = 500ms$, $S_c = 2MB$, $E_c = 3.0mJ$, and L_c=25). H-Gen perform the two-stage optimization (see Algorithm 1) to generate the H-CNN. We chose FER-2013 (containing 28,709 training facial images, 3589 private test facial images, and 3589 public test facial images) as the dataset, which provides cropped facial images with 48×48 pixels.

In the inference stage, the H-CNN on the mobile device will perform tasks constantly and upload low-confidence samples in challenging scenarios to the H-Gen server. The H-Gen server completes the data annotation through the crowdsourcing platform. The crowdsourcing platform will check the participant's status and only assign the annotation tasks to online participants. We employ four students to install the crowdsourcing application (*i.e.*, WeSense) [39] on their mobile phones and discover, accept, and complete the annotation tasks. The annotations are stored in an extra human knowledge base. When new human knowledge accumulates to a preset threshold(*e.g.* 200 for H-CNN), H-Gen will retrain H-CNN. H-Gen will use new labeled data to fine-tune H-CNN weights and update the new H-CNN to mobile devices. We test the H-CNN's performance in terms of accuracy, latency, storage, and energy cost under three challenging scenarios (*i.e.*, angle changing, moving scenes, and low lighting).

Table 7 summarizes the results. In the angle-changing scenario, 10 out of the 120 frames are for human annotation, and the average latency is 494.5ms (including network latency of 29.4ms) in 2s. The accuracy is 70.4%. In the moving scenes scenario, 14 out of the 120 frames are for human annotation, and the average latency is 498.5ms(including network latency of 32.6ms) in 2s. The accuracy is 70.1%. In the low lighting scenario, 8 out of the 120 frames are for human annotation, and the average latency is 488.5ms(including network latency of 20.5ms) in 2s. The accuracy is 70.9%.

8 CONCLUSION

We propose H-Gen, an automatic model specialization and compression framework for human-in-the-loop DNNs (H-DNNs). It is the first proposal to incorporate human participation as a new hyperparameter into the design space of efficient H-DNN generation for better performance-resource trade-off. H-Gen formulates the

search space, metric calculation, and search strategy to build a NAS-style H-DNN hyperparameter tuning framework. We also propose human participation mechanisms for three common DNN architectures to showcase the feasibility of H-Gen. Evaluations on twelve challenging mobile tasks over three H-DNN architectures demonstrate the effectiveness of H-Gen. We envision H-Gen as one step closer to automatic, intelligent, and ubiquitous human-in-the-loop machine learning.

In the future, we would like to improve H-Gen in the following aspects. *(i)* Explore generic rationales and optimal mechanisms for human annotation. This paper only serves as a feasibility study with three showcases (H-CNN, H-YOLO, H-Faster RCNN) to integrate human annotations into the DNN development chain, which results in architecture-specific designs. It would be interesting to investigate principles and rationales that contribute to unified and optimal designs of human annotations for DNNs. *(ii)* Devise crowdsourcing incentives and mechanisms suited for human-in-the-loop DNNs. We simplify our design by assuming that human annotators are always available to provide high-quality annotations because our design only requires a small portion of samples to be annotated. However, designing crowdsourcing mechanisms dedicated to human-in-the-loop DNNs can further improve the quality and the latency of human annotation on these samples. Practical issues such as incentive mechanisms, task assignments, quality control, and user interfaces would also contribute to the large-scale deployment of H-Gen. *(iii)* Integrate more performance metrics related to mobile devices. The current version of H-Gen implicitly considers the computation resources of mobile devices in the latency. An immediate improvement is explicitly integrating metrics such as MACs and Ops into the optimization framework. We can also consider finer-grained human labor assessments such as match degree and reputation level.

ACKNOWLEDGMENTS

This work was supported in part by the National Key R&D Program of China (No. 2021YFB2900100) and the National Natural Science Foundation of China (No. 61960206008, No. 62102317, No. 62025205, No. 62032020). The authors also thank the anonymous reviewers for their valuable comments that has made the work stronger.

REFERENCES

[1] Alireza Abedin, Mahsa Ehsanpour, Qinfeng Shi, Hamid Rezatofighi, and Damith C Ranasinghe. 2021. Attend and Discriminate: Beyond the State-of-the-Art for Human Activity Recognition Using Wearable Sensors. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 5, 1 (2021), 1–22.

[2] Bishwo Adhikari and Heikki Huttunen. 2021. Iterative bounding box annotation for object detection. In *Proceedings of the IEEE International Conference on Pattern Recognition*. IEEE, Piscataway, NJ, USA, 4040–4046.

[3] Amazon. 2019. Amazon mechanical turk. https://www.mturk.com/.

[4] Ines Arous, Jie Yang, Mourad Khayati, and Philippe Cudré-Mauroux. 2020. Opencrowd: A human-ai collaborative approach for finding social influencers via open-ended answers aggregation. In *Proceedings of The Web Conference*. ACM, New York, NY, USA, 1851–1862.

[5] Luca Arrotta, Gabriele Civitarese, and Claudio Bettini. 2022. DeXAR: Deep Explainable Sensor-Based Activity Recognition in Smart-Home Environments. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 6, 1 (2022), 1–30.

[6] Sourav Bhattacharya and Nicholas D Lane. 2016. Sparsification and separation of deep learning layers for constrained resource inference on wearables. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*. ACM, New York, NY, USA, 176–189.

[7] Steven L Brunton and J Nathan Kutz. 2022. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, Cambridge, UK.

[8] Chengliang Chai, Lei Cao, Guoliang Li, Jian Li, Yuyu Luo, and Samuel Madden. 2020. Human-in-the-loop outlier detection. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, New York, NY, USA, 19–33.

[9] Soravit Changpinyo, Mark Sandler, and Andrey Zhmoginov. 2017. The power of sparsity in convolutional neural networks. arXiv preprint arXiv:1702.06257.

[10] Jiwoong Choi, Ismail Elezi, Hyuk-Jae Lee, Clement Farabet, and Jose M Alvarez. 2021. Active learning for deep object detection via probabilistic modeling. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. IEEE, Piscataway, NJ, USA, 10264–10273.

[11] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, et al. 2019. Chamnet: Towards efficient network design through platform-aware model adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, Piscataway, NJ, USA, 11398–11407.

[12] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. 2020. Model compression and hardware acceleration for neural networks: a comprehensive survey. *Proc. IEEE* 108, 4 (2020), 485–532.

[13] Samuel Dodge and Lina Karam. 2019. Human and DNN classification performance on images with quality distortions: A comparative study. *ACM Transactions on Applied Perception (TAP)* 16, 2 (2019), 1–17.

[14] Jin-Dong Dong, An-Chieh Cheng, Da-Cheng Juan, Wei Wei, and Min Sun. 2018. Ppp-net: Platform-aware progressive search for pareto-optimal neural architectures. International Conference on Learning Representations Workshops.

[15] Xuanyi Dong, Mingxing Tan, Adams Wei Yu, Daiyi Peng, Bogdan Gabrys, and Quoc V Le. 2021. AutoHAS: Efficient hyperparameter and architecture search. International Conference on Learning Representations Workshops.

[16] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural architecture search: A survey. *The Journal of Machine Learning Research* 20, 1 (2019), 1997–2017.

[17] Mark Everingham, Andrew Zisserman, Christopher KI Williams, Luc Van Gool, Moray Allan, Christopher M Bishop, Olivier Chapelle, Navneet Dalal, Thomas Deselaers, Gyuri Dorkó, et al. 2008. The PASCAL visual object classes challenge 2007 (VOC2007) results.

[18] Anna Lisa Gentile, Daniel Gruhl, Petar Ristoski, and Steve Welch. 2019. Explore and exploit. Dictionary expansion with human-in-the-loop. In *European Semantic Web Conference*. Springer, Berlin, Germany, 131–145.

[19] Ross Girshick. 2015. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*. IEEE, Piscataway, NJ, USA, 1440–1448.

[20] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. 2017. On calibration of modern neural networks. In *Proceedings of the ACM International Conference on Machine Learning*. ACM, New York, NY, USA, 1321–1330.

[21] Sairam Gurajada, Lucian Popa, Kun Qian, and Prithviraj Sen. 2019. Learning-based methods with human-in-the-loop for entity resolution. In *Proceedings of the ACM on Conference on Information and Knowledge Management*. ACM, New York, NY, USA, 2969–2970.

[22] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European conference on computer vision (ECCV)*. Springer, Berlin, Germany, 784–800.

[23] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. IEEE, Piscataway, NJ, USA, 1389–1397.

[24] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. 2015. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531.

[25] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

[26] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. arXiv preprint arXiv:1602.07360.

[27] Hongbo Jiang, Hangcheng Cao, Daibo Liu, Jie Xiong, and Zhichao Cao. 2020. Smileauth: Using dental edge biometrics for user authentication on smartphones. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 3 (2020), 1–24.

[28] Bongjun Kim and Bryan Pardo. 2018. A human-in-the-loop system for sound event detection and annotation. *ACM Transactions on Interactive Intelligent Systems (TiiS)* 8, 2 (2018), 1–23.

[29] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. 2016. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. ACM, New York, NY, USA, 1–12.

[30] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[31] Qiaozhe Li, Jiahui Zhang, Xin Zhao, and Kaiqi Huang. 2021. Can DNN Detectors Compete Against Human Vision in Object Detection Task?. In *Proceedings of Chinese Conference on Pattern Recognition and Computer Vision*. Springer, Berlin, Germany, 542–553.

[32] Min Lin, Qiang Chen, and Shuicheng Yan. 2013. Network in network. International Conference on Learning Representations.

[33] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. 2015. Sparse convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. IEEE, Piscataway, NJ, USA, 806–814.

[34] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2018. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision*. Springer, Berlin, Germany, 19–34.

[35] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. DARTS: Differentiable Architecture Search. International Conference on Learning Representations.

[36] Sicong Liu, Junzhao Du, Kaiming Nan, Zimu Zhou, Hui Liu, Zhangyang Wang, and Yingyan Lin. 2020. AdaDeep: a usage-driven, automated deep model compression framework for enabling ubiquitous intelligent mobiles. *IEEE Transactions on Mobile Computing* 20, 12 (2020), 3282–3297.

[37] Sicong Liu, Bin Guo, Ke Ma, Zhiwen Yu, and Junzhao Du. 2021. AdaSpring: Context-adaptive and Runtime-evolutionary Deep Model Compression for Mobile Applications. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 5, 1 (2021), 1–22.

[38] Sicong Liu, Yingyan Lin, Zimu Zhou, Kaiming Nan, Hui Liu, and Junzhao Du. 2018. On-demand deep model compression for mobile devices: A usage-driven model selection framework. In *Proceedings of the Annual International Conference on Mobile Systems, Applications, and Services*. ACM, New York, NY, USA, 389–400.

[39] Yimeng Liu, Zhiwen Yu, Bin Guo, Qi Han, Jiangbin Su, and Jiahao Liao. 2020. CrowdOS: A ubiquitous operating system for crowdsourcing and mobile crowd sensing. *IEEE Transactions on Mobile Computing* (2020).

[40] Zimo Liu, Jingya Wang, Shaogang Gong, Huchuan Lu, and Dacheng Tao. 2019. Deep reinforcement active learning for human-in-the-loop person re-identification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. IEEE, Piscataway, NJ, USA, 6122–6131.

[41] Qinghao Meng, Wenguan Wang, Tianfei Zhou, Jianbing Shen, Yunde Jia, and Luc Van Gool. 2021. Towards a weakly supervised framework for 3d point cloud object detection and annotation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 0, 0 (2021), 1–1.

[42] Szymon Migacz. 2017. 8-bit inference with tensorrt," 2017. GPU Technology Conference.

[43] 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.

[44] Akshay Uttama Nambi, Adtiya Virmani, and Venkata N Padmanabhan. 2018. FarSight: a smartphone-based vehicle ranging system. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 4 (2018), 1–22.

[45] Andrew Ng et al. 2011. Sparse autoencoder. *CS294A Lecture notes* 72, 2011 (2011), 1–19.

[46] Chi Cuong Nguyen, Giang Son Tran, Jean-Christophe Burie, Thi Phuong Nghiem, et al. 2021. Pulmonary Nodule Detection Based on Faster R-CNN With Adaptive Anchor Box. *IEEE Access* 9 (2021), 154740–154751.

[47] Antonio Polino, Razvan Pascanu, and Dan Alistarh. 2018. Model compression via distillation and quantization. International Conference on Learning Representations.

[48] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, Piscataway, NJ, USA, 779–788.

[49] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in Neural Information Processing Systems* 28 (2015), 1–9.

[50] Liu Sicong, Zhou Zimu, Du Junzhao, Shangguan Longfei, Jun Han, and Xin Wang. 2017. Ubiear: Bringing location-independent sound awareness to the hard-of-hearing people with smartphones. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 2 (2017), 1–21.

[51] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic policy gradient algorithms. In *International conference on machine learning*. ACM, New York, NY, USA, 387–395.

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

[53] Pravendra Singh, Vinay Kumar Verma, Piyush Rai, and Vinay P Namboodiri. 2019. Play and prune: Adaptive filter pruning for deep model compression. arXiv preprint arXiv:1905.04446.

[54] Yunpeng Song and Zhongmin Cai. 2022. Integrating Handcrafted Features with Deep Representations for Smartphone Authentication. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 6, 1 (2022), 1–27.

[55] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, Piscataway, NJ, USA, 2820–2828.

[56] Jilin Tu, Ana Del Amo, Yi Xu, Li Guari, Mingching Chang, and Thomas Sebastian. 2012. A fuzzy bounding box merging technique for moving object detection. In *2012 Annual Meeting of the North American Fuzzy Information Processing Society (NAFIPS)*. IEEE, 1–6.

[57] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. 2016. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*. ACM, New York, NY, USA, 1995–2003.

[58] Garrett Warnell, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. 2018. Deep tamer: Interactive agent shaping in high-dimensional state spaces. In *Proceedings of the AAAI conference on artificial intelligence*. AAAI, Palo Alto, CA, USA, 1545–1553.

[59] Xingjiao Wu, Luwei Xiao, Yixuan Sun, Junhang Zhang, Tianlong Ma, and Liang He. 2021. A Survey of Human-in-the-loop for Machine Learning. arXiv preprint arXiv:2108.00941.

[60] Wentao Xie, Qian Zhang, and Jin Zhang. 2021. Acoustic-based Upper Facial Action Recognition for Smart Eyewear. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 5, 2 (2021), 1–28.

[61] Yiqun Xie, Jiannan Cai, Rahul Bhojwani, Shashi Shekhar, and Joseph Knight. 2020. A locally-constrained yolo framework for detecting small and densely-distributed building footprints. *International Journal of Geographical Information Science* 34, 4 (2020), 777–801.

[62] Doris Xin, Litian Ma, Jialin Liu, Stephen Macke, Shuchen Song, and Aditya Parameswaran. 2018. Accelerating human-in-the-loop machine learning: Challenges and opportunities. In *Proceedings of Workshop on Data Management for End-to-End Machine Learning*. ACM, New York, NY, USA, 1–4.

[63] Mengwei Xu, Feng Qian, and Saumay Pushp. 2017. Enabling cooperative inference of deep learning on wearables and smartphones. arXiv preprint arXiv:1712.03073.

[64] Fan Yang, Zhiwen Yu, Liming Chen, Jiaxi Gu, Qingyang Li, and Bin Guo. 2021. Human-machine cooperative video anomaly detection. *Proceedings of the ACM on Human-Computer Interaction* 4, CSCW3 (2021), 1–18.

[65] Zhican Yang, Chun Yu, Fengshi Zheng, and Yuanchun Shi. 2019. ProxiTalk: Activate Speech Input by Bringing Smartphone to the Mouth. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 3, 3 (2019), 1–25.

[66] Shuochao Yao, Yiran Zhao, Aston Zhang, Lu Su, and Tarek Abdelzaher. 2017. Deepiot: Compressing deep neural network structures for sensing systems with a compressor-critic framework. In *Proceedings of the ACM Conference on Embedded Network Sensor Systems*. ACM, New York, NY, USA, 1–14.

[67] Dewei Yi, Jinya Su, and Wen-Hua Chen. 2021. Probabilistic faster R-CNN with stochastic region proposing: Towards object detection and recognition in remote sensing imagery. *Neurocomputing* 459 (2021), 290–301.

[68] Dewei Yi, Jinya Su, Cunjia Liu, and Wen-Hua Chen. 2017. Personalized driver workload inference by learning from vehicle related measurements. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 49, 1 (2017), 159–168.

[69] Zhiwen Yu, Qingyang Li, Fan Yang, and Bin Guo. 2021. Human–machine computing. *CCF Transactions on Pervasive Computing and Interaction* 3, 1 (2021), 1–12.

[70] Fabio Massimo Zanzotto. 2019. Human-in-the-loop artificial intelligence. *Journal of Artificial Intelligence Research* 64 (2019), 243–252.

[71] Arber Zela, Aaron Klein, Stefan Falkner, and Frank Hutter. 2018. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. arXiv preprint arXiv:1807.06906.

[72] Daniel Yue Zhang, Yifeng Huang, Yang Zhang, and Dong Wang. 2020. Crowd-assisted disaster scene assessment with human-ai interactive attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI, Palo Alto, CA, USA, 2717–2724.

[73] Lvmin Zhang, Xinrui Wang, Qingnan Fan, Yi Ji, and Chunping Liu. 2021. Generating manga from illustrations via mimicking manga creation workflow. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, Piscataway, NJ, USA, 5642–5651.

[74] Peng Zhang, Jianye Hao, Weixun Wang, Hongyao Tang, Yi Ma, Yihai Duan, and Yan Zheng. 2021. KoGuN: Accelerating Deep Reinforcement Learning via Integrating Human Suboptimal Knowledge. In *Proceedings of International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, Burlington, MA, USA, 2291–2297.

[75] Ruohan Zhang, Faraz Torabi, Lin Guan, Dana H Ballard, and Peter Stone. 2019. Leveraging Human Guidance for Deep Reinforcement Learning Tasks. In *Proceedings of International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, Burlington, MA, USA, 6339–6346.

[76] Xiaozhao Zhao, Yuexian Hou, Dawei Song, and Wenjie Li. 2017. A confident information first principle for parameter reduction and model selection of boltzmann machines. *IEEE Transactions on Neural Networks and Learning Systems* 29, 5 (2017), 1608–1621.

[77] Xiawu Zheng, Rongrong Ji, Lang Tang, Baochang Zhang, Jianzhuang Liu, and Qi Tian. 2019. Multinomial distribution learning for effective neural architecture search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. IEEE, Piscataway, NJ, USA, 1304–1313.

[78] Yan Zhuang, Guoliang Li, Zhuojian Zhong, and Jianhua Feng. 2017. Hike: A hybrid human-machine method for entity alignment in large-scale knowledge bases. In *Proceedings of the ACM on Conference on Information and Knowledge Management*. ACM, New York, NY, USA, 1917–1926.

[79] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, Piscataway, NJ, USA, 8697–8710.

APPENDIX

In Equation (15), the coefficients δ_1 and δ_2 are defined by user demands as input for H-Gen. H-Gen automatically assigns the coefficient $\delta_3, \delta_4, and \delta_5$ according to different resource constraints. The labor cost factor δ_5 is directly determined by the maximum labor cost acceptable to the user using function $\delta_5 = \frac{0.1-L_c}{0.1}$. The lower the labor cost constraint is, the greater the reward coefficient for saving labor costs is. 0.1 is the maximum L_c that the system allows for input (preset in H-Gen). Following AdaDeep, δ_4 is set as $max\{\frac{3800-E_{battery}}{3800}, 0.5\}$. And $\delta_3 = 1 - \delta_5 - \delta_4$.

Table 8. Summary of major notations.

Notation	Explanation
A, T, E, S, L	accuracy, latency, energy, storage and labour cost of H-DNN
A_d, T_d	accuracy and latency requirements
E_c, S_c, L_c	energy, storage, labour cost constraints
$c_{H-CNN}, c_{H-YOLO}, c_{H-Faster-RCNN}$	inference confidence of H-CNN, H-YOLO, H-Faster-RCNN
$L^0_{H-CNN}, L^0_{H-YOLO}, L^0_{H-Faster-RCNN}$	human labour cost per sample of H-CNN, H-YOLO, H-Faster-RCNN
$anno_i$, $\{anno_n(w, h, x, y, c)\}$, $\{anno_n(w, h, x, y)\}$	human annotation per participant of H-CNN, H-YOLO, H-Faster-RCNN
w, h, x, y	width, height, x-coordinate, y-coordinate of annotation box
$\bar{w}, \bar{h}, \bar{x}, \bar{y}$	mean width, height, x-coordinate, y-coordinate of annotation box
δ_1, δ_2	coefficient to balance A and T
$\delta_3, \delta_4, \delta_5$	coefficient to balance S, E and L
υ	hyperparameter for model compression
τ	threshold to determine amount of human participation
T_{infer}, T_{extra}	DNN inference time, extra latency by human participation per batch
$\mathcal{S}_f, \mathcal{S}_p$	storage for activations and weights
$\mathcal{S}_a, \mathcal{S}_w$	total # activations and weights in DNN
B_a, B_w	# bits per activation and weight
η	energy cost per MAC
MAC_s	# multiply-accumulat of DNN
L_{actual}, L_{max}	actual and maximum acceptable labour cost per batch inference
n, N	# boxes annotated per participant, # samples to be annotated per batch
$\mathcal{M}$	batch size
$\mathcal{I}(.), Y(.),$	indicator functions
$c_{c_i}, c_{c_{r_i}}$	classification and regression confidence in grid i
W, H, G	width, height, # grid of an input image
G_{x_i}, G_{y_i}	convolution in the horizontal and vertical direction of each image pixel
o^i_c, o^j_f	compression techniques for $conv$ and fc layer
m	# human participants
k	# categories in classification or object detection
Φ	search space of υ
λ	a factor in H-CNN
c	class of human annotated object
μ_i	mixture weight for Gaussian component
Ω	optimal component number in H-Faster-RCNN
Σ	covariance matrix of size and position of human annotation areas
σ_{SM}	softmax operator
z	logit k-dimensional vector
$\mathcal{T}$	a learnable parameter in H-CNN
l, h_l, w_l, c_l	index, height, width, and channels of input features to DNN layer l
$\omega, \omega^{actor}, \omega^{critic}$	parameters of DQN, DDPG actor and DDPG critic
O, O'	action set of DQN and DDPG