Partial Weight Adaptation for Robust DNN Inference

Xiufeng Xie
Hewlett Packard Labs
xiufeng.xie@hpe.com

Kyu-Han Kim
Hewlett Packard Labs
kyu-han.kim@hpe.com

Abstract

Mainstream video analytics uses a pre-trained DNN model with an assumption that inference input and training data follow the same probability distribution. However, this assumption does not always hold in the wild: autonomous vehicles may capture video with varying brightness; unstable wireless bandwidth calls for adaptive bitrate streaming of video; and, inference servers may serve inputs from heterogeneous IoT devices/cameras. In such situations, the level of input distortion changes rapidly, thus reshaping the probability distribution of the input.

We present GearNN*, an adaptive inference architecture that accommodates DNN inputs with varying distortions.* GearNN *employs an optimization algorithm to identify a tiny set of "distortion-sensitive" DNN parameters, given a memory budget. Based on the distortion level of the input,* GearNN *then adapts only the distortion-sensitive parameters, while reusing the rest of DNN parameters across all input qualities. In our evaluation of DNN inference with dynamic input distortions,* GearNN *improves the accuracy (mIoU) by an average of 18.12% over a DNN trained with the undistorted dataset and 4.84% over stability training from Google, with only 1.8% extra memory overhead.*

1. Introduction

Video analytics solutions typically use a DNN with pre-trained weights for inference, assuming consistent probability distribution between the training and test dataset. Unfortunately, the inputs to DNN inference might have various distortions that alter the probability distribution and harm DNN performance in practice. An autonomous vehicle may drive in and out of shades, causing abrupt brightness change in the captured video; a drone may change a compression ratio of video frames while streaming to the inference server based on wireless link bandwidth; edge servers may need to process data from IoT devices with heterogeneous camera hardware and compression strategies. In these scenarios of input distortions with high dynamic range, existing solutions that rely on a DNN with constant pre-trained weights suffer from severe accuracy loss [23]. We observe up to a 58% accuracy loss in our experiments (§5).

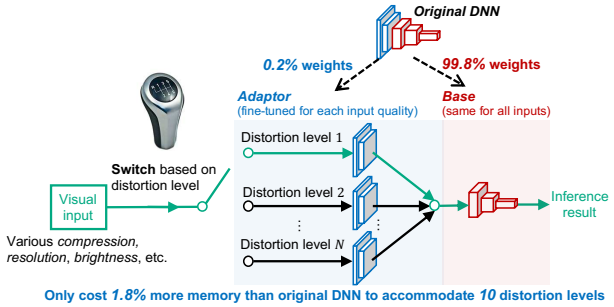


Figure 1: GearNN, an adaptive inference architecture (This is a simplified illustration, DNN layers in the *adaptor* and those in the *base* can be interleaved with each other).

One workaround to handle input with unstable distortion level is to train a DNN for each possible distortion level by augmenting the training data to match that particular distortion level, and then switch between them following the distortion level of the current input. However, there are enormous distortion levels (*e.g.*, JPEG has 100 quality levels), concurrently running so many DNNs is infeasible, due to limited memory. Swapping DNNs between disk and memory causes huge latency, and thus, is impractical.

This paper proposes GearNN, an adaptive DNN inference architecture to accommodate real-world inputs with various distortions, without sacrificing memory efficiency. GearNN only adapts a tiny portion (*e.g.*, 0.2% of the DNN size in §4) of the DNN weights that are "distortion sensitive" (called *adaptor*) to fit the distortion level of the instantaneous input, while reusing the majority of weights (called *base*) across all inputs. In this way, the adaptation leads to high inference accuracy, while reusing most weights guarantees memory efficiency and scalability. We name our design GearNN: like the gearbox that helps one engine handle different car speeds, GearNN helps a single DNN *base* to accommodate a wide range of input distortions.

The GearNN workflow can be summarized as follows:

(i) Identifying distortion-sensitive weights offline. Given a DNN pre-trained with undistorted training dataset,

GearNN first fine-tunes this DNN to multiple versions, each with training data of a particular distortion level. Next, by comparing the original DNN and the fine-tuned versions, GearNN runs an optimization problem (§3.2.2) to identify a set of distortion-sensitive DNN weights, under a constraint of additional memory budget.

(ii) Partial DNN fine-tuning offline. GearNN then partially fine-tunes the DNN for each pre-defined distortion level, by only updating the distortion-sensitive weights (*i.e.*, *adaptor*), while freezing the rest of the pre-trained DNN weights (*i.e.*, *base*). This step yields multiple *adaptors*, each for a particular distortion level.

(iii) Partial DNN adaptation online. With multiple fine-tuned small *adaptors* and a single copy of the *base* loaded in memory, GearNN switches between the *adaptors*, following a current input distortion level (like compression level), while reusing the *base* across all possible inputs.

We have prototyped GearNN on top of popular DNN models like DRN [21, 22] and Mask R-CNN [6] using PyTorch, and performed an extensive evaluation with semantic segmentation and detection tasks (§5). Our evaluation shows that GearNN enables robust DNN inference under various input distortion levels, while retaining memory efficiency. More specifically, GearNN achieves up to 18.12% higher average inference accuracy over the original DNN trained with undistorted data. GearNN also outperforms other alternatives such as stability training from Google or fine-tuning a DNN using a dataset with mixed distortion levels, and it consumes only 1.8% more memory over such single DNN solutions to accommodate 10 distortion levels. Meanwhile, compared to switching between multiple DNNs, GearNN reduces memory consumption by 88.7%, while achieving a similar level of accuracy.

Our contributions can be summarized as follows:

- We propose GearNN, a general technique to improve the tolerance of DNN models to input distortions with high dynamic range, without compromising the memory efficiency. We validate GearNN over state-of-the-art DNNs, where it outperforms existing solutions.
- GearNN formulates an optimization problem that selects a tiny set of DNN weights to be adapted, under a given constraint of memory consumption.
- GearNN is the first to enable robust video analytics on adaptive bitrate (ABR) streaming, following the trend of modern video streaming.
- GearNN can quickly customize (only takes 4 training epochs in our prototype) any pre-trained DNN to fit the desired dynamic range of input distortion, offering flexibility in deployment.

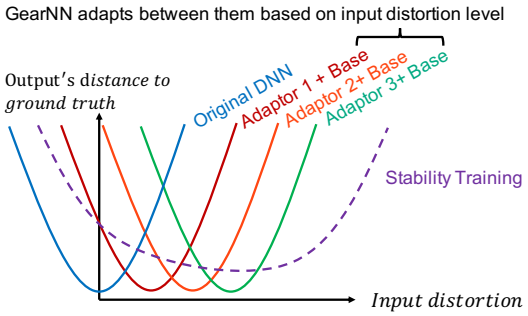


Figure 2: An adaptive DNN can better serve a wide dynamic range of input distortions than a constant DNN.

2. Related Work

Adaptive neural network. Existing work [1, 18] discusses adapting the neural network architecture based on the instantaneous input. However, these solutions focus only on accelerating the inference by early exit [1] or skipping convolution layers [18]. In contrast, GearNN aims to improve the DNN robustness under diverse & dynamic input distortions without compromising memory consumption. Besides, GearNN adapts DNN weight values, instead of network architecture, which guarantees fast adaptation and backward compatibility.

Improve training for better robustness. Some existing work proposes to improve the training phase to make DNNs more robust to small input perturbations. *Stability training* [23] from Google uses a modified training architecture that flattens the input-output mapping in a small neighborhood of the input image. However, for perturbation with a broad range, it is difficult to flatten one input-output mapping, as shown in Fig. 2. Hence, *stability training* fails to tolerate various input distortions. On the other hand, GearNN processes the input with different distortion levels by using different input-output mappings, and thus, it is more tolerable under various input distortions.

DNN pruning. DNN pruning [4, 11, 8] identifies and removes trivial neurons to reduce the model size and computational cost. This approach looks similar to our solution at first glance but is fundamentally different. GearNN focuses on the "distortion-sensitive" weights rather than the neurons that are important to the output. Some DNN weights can be insensitive to the input distortion but remain vital to the output. In other words, GearNN can reuse such weights across different input distortion levels, but the DNN cannot work correctly after pruning such weights. Besides, DNN pruning uses a pruned constant DNN for inference, while GearNN adapts (part of) DNN online during the inference.

Transfer learning. Due to the high cost to build a large training dataset, transfer learning [12, 16, 20] is proposed to customize a pre-trained model on one task to fit another task. Although GearNN follows the principle of

transfer learning – customizing the pre-trained model to fit another task of same input type but different distortion levels – it is drastically different from existing practice of transfer learning: *(i)* GearNN actively identifies the "distortion-sensitive" weights and only fine-tunes such weights. In contrast, typical transfer learning fine-tunes either only the fully-connected layer or all DNN weights. *(ii)* GearNN enables partial weight adaptation in runtime inference, whereas typical transfer learning still uses a pre-trained DNN for inference.

Adaptive bitrate streaming. Adaptive bitrate (ABR) streaming quickly gains popularity in recent years [9, 13, 15]. By adapting the media stream quality in real-time following the dynamic network bandwidth, ABR streaming achieves both the high throughput and low latency. Unfortunately, existing DNN-based inference solutions mostly use a single DNN and fail to accommodate the dynamic quality levels on the ABR streams. In contrast, GearNN can adapt the DNN to accommodate dynamic streaming quality.

3. GearNN: Partial weight adaptation

In this section, we first analyze the effect of fine-tuning a DNN with distorted training data (§3.1), and then reveal our observation that the majority of DNN weights have *minor* changes after fine-tuning with distorted data (§3.2). Finally, we elaborate on the GearNN design, featuring partial DNN adaptation motivated by our observation (§3.3).

3.1. What happens when fine-tuning a DNN?

3.1.1 Frequency-domain features of visual distortions

Since human eyes are insensitive to the high-frequency components (in the spatial-frequency domain) of the visual input, encoders like JPEG and H.264 compress such high-freq. components aggressively. Similarly, after reducing the image brightness, human eyes still perceive object outlines (low-freq. components), but not textures (high-freq. components). In other words, typical visual distortions in practice essentially add noise to the high-freq. components.

3.1.2 DNN's frequency response after fine-tuning

Inspired by human eyes' spatial frequency response, we measure DNNs' spatial frequency response to understand how DNNs respond to the noise caused by visual distortions (§3.1.1). As shown in Fig. 3, we prepend an Inverse Discrete Fourier Transform (IDCT) module to the typical DNN training framework, then the frequency components (DCT coefficients) of the input become the leaf nodes, and we can perform backward propagation to obtain the gradient of loss *w.r.t.* every input frequency component. A higher gradient amplitude of a frequency component indicates that the DNN is more sensitive to input noise on this frequency,

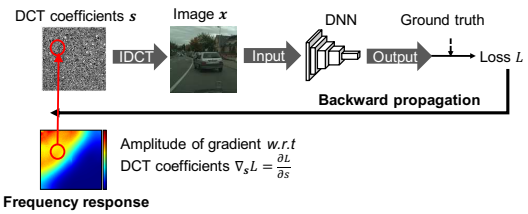


Figure 3: Using the gradient of loss to model the DNN's freq.-domain perceptual sensitivity (*frequency response*).

and we define the gradient map of all input frequency components as the DNN's *frequency response* [19].

We then compare the *frequency responses* of the original DNN and the DNN fine-tuned with distorted training data. We repeat the comparison for 4 types of distortions: *(i)* H.264 compression with a quality (CRF) of 24; *(ii)* JPEG compression with a quality of 10; *(iii)* Underexposure with 10% brightness of the original image; *(iv)* Data augmentation that mixes H.264 frames with different qualities (CRF from 15 to 24) in one training set. Fig. 4 shows the measured DNN frequency responses. We see that, compared to the original DNN, the DNNs fine-tuned with distorted training data become less sensitive to the high-frequency components in all tested cases, *i.e.*, they learn to avoid looking at the noisy high-frequency components for better robustness.

3.2. Distortion-sensitive DNN weights

Fine-tuning a DNN is updating its weights to fit a new training set. In §3.1, we reveal that fine-tuning a DNN with a distorted dataset is equivalent to reshaping its *frequency response* so that it filters out the noise from the distortion. In this section, we dig further to understand which DNN weights play more critical roles than other ones in reshaping the *frequency response*. We define such important weights as "distortion-sensitive". It is worth noting that the "distortion-sensitivity" is different from the importance of neurons in DNN pruning. Some weights can be vital for the inference and should not be pruned. However, they can meanwhile be insensitive to the input distortion, *i.e.*, their values do not change much after fine-tuning with distorted training data – there is no need to fine-tune them because the inference output is insensitive to the small changes of the DNN weights as shown in existing works [5, 3, 10].

3.2.1 Modeling DNN weights' sensitivity to distortions

We first fine-tune a pre-trained DNN $\mathcal{D}$ by using distorted training datasets, and compare the resulting weights with the original model. Let K denote the number of layers in $\mathcal{D}$, and $\boldsymbol{l}_i$ denotes each layer, which contains N_i weights, then we have $\boldsymbol{l}_i = \{p_i^1, p_i^2 \dots p_i^{N_i}\}$. Let p_i^j denote the j-th weight of the layer $\boldsymbol{l}_i$ in the original pre-trained DNN, and $f_q(\cdot)$ denotes the fine-tuning process with a certain distor-

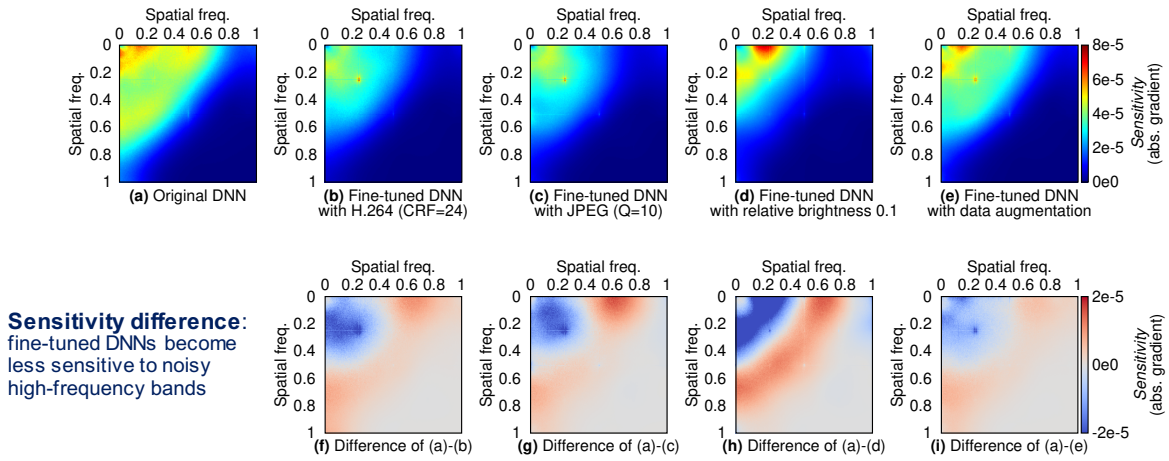
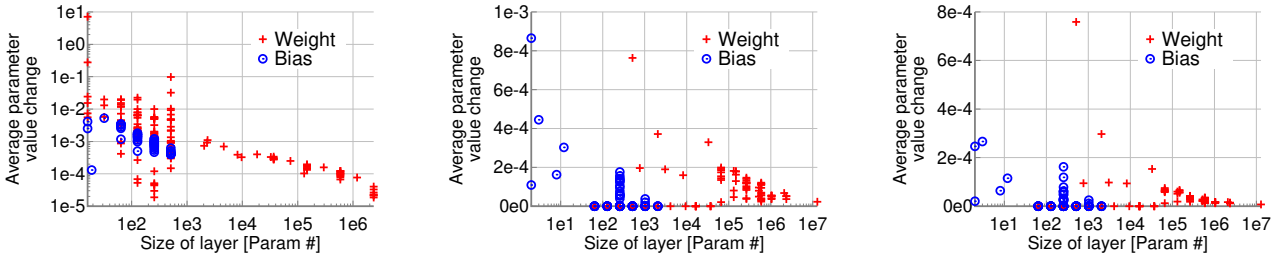


Figure 4: Comparing the DCT spectral sensitivity (*frequency response*) of the original and fine-tuned `DRN-D-38`. We use different color palettes for the sensitivity in the 1st row and the sensitivity difference (can be negative) in the 2nd row.



(a) `DRN-D-38` fine-tuned with H.264 CRF=23. (b) `Mask R-CNN` fine-tuned with JPEG Q=10. (c) `Mask R-CNN` fine-tuned with relative brightness 0.2.

Figure 5: Per-layer average weight value change caused by fine-tuning.

tion level q. Then, $f_q(p_i^j)$ is the corresponding weight of the fine-tuned DNN, and we can compute the average change of weight values in layer $\boldsymbol{l}_i$, caused by fine-tuning, as:

$$v_i^q = \frac{1}{N_i} \sum_{j=1}^{N_i} \|p_i^j - f_q(p_i^j)\| \tag{1}$$

The layers with a high v_i^q value yield significant change of weight values, when fine-tuned to fit the distortion level q, which means they are sensitive to the distortion level. Therefore, we define v_i^q as the *distortion sensitivity* of layer $\boldsymbol{l}_i$. Following Eq. (1), we measure the layer-level weight change caused by the fine-tuning for three cases: `DRN-D-38` fine-tuned with H.264 of quality level (CRF) 23 in Fig 5a; `Mask R-CNN` fine-tuned with JPEG of quality level (Q) 10 in Fig 5b; and, `Mask R-CNN` fine-tuned with dimmed images of relative brightness 0.2 in Fig 5c.

In Fig. 5a, the sum size of the `DRN` parameters (including both the weights & biases. We use the term "weights" to denote "DNN parameters including both weights & biases" elsewhere for simplicity) that changed more than 2×10^{-4} after fine-tuning only accounts for 1.44% of the model size. Similarly, only 0.08% and 0.0058% of the `Mask R-CNN` weights changed more than 2×10^{-4} in Fig. 5b and Fig. 5c. We thus have a key observation: only a tiny portion of DNN weights have non-negligible changes (*e.g.*, $> 2 \times 10^{-4}$) after fine-tuning with distorted data. *In order to fit DNN to the distorted inputs, we can reshape the frequency response of a DNN by changing such a tiny portion of DNN weights.*

3.2.2 Choosing subset of DNN weights to fine-tune

Since only a tiny portion of DNN weights have non-negligible changes after fine-tuning with distorted data, given a constraint on memory usage, `GearNN` can select and then fine-tune only these "distortion-sensitive" weights. We formulate this as a knapsack problem in Eq. (2), where the "cost" of layer $\boldsymbol{l}_i$ is its parameter number N_i and the "value" of layer $\boldsymbol{l}_i$ is its *distortion sensitivity* defined in Eq. (1). More specifically, we need to select a list S of DNN layers to maximize the total "value" (*distortion sensitivity*) $\sum_{i \in S} v_i^q$, under the constraint that the total "cost" (memory usage) $\sum_{i \in S} N_i$ is within a user-defined bound M.

$$\max_S \sum\nolimits_{\boldsymbol{i} \in S} v_i^q \quad \text{s.t.} \quad \sum\nolimits_{\boldsymbol{i} \in S} N_i \leq M \tag{2}$$

After obtaining the optimal list S of layers, we fine-tune the corresponding DNN portion $\mathcal{A}_q = \{l_i : i \in S\}$ (we call it

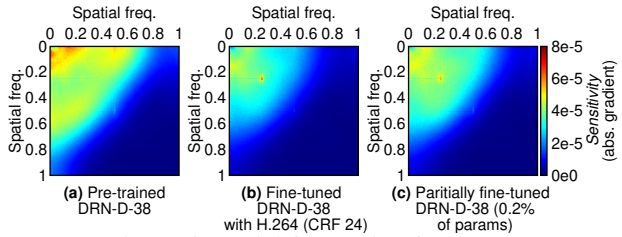


Figure 6: Partial fine-tuning for DRN.

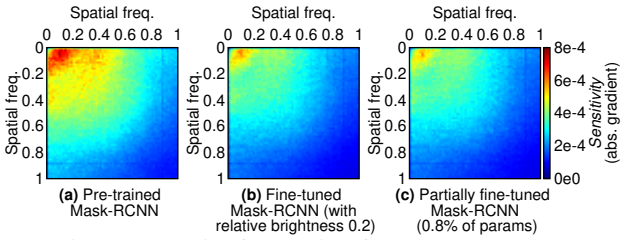


Figure 7: Partial fine-tuning for Mask R-CNN.

the *adaptor*) with the dataset of quality q while keeping the rest of the DNN (we call it the *base* $\mathcal{B}_q = \mathcal{D} \setminus \mathcal{A}_q$) frozen. We denote this partial fine-tuning step as $\mathcal{A}_q^* = f_q(\mathcal{A}_q)$.

3.2.3 Frequency response of partially fine-tuned DNN

We further compare the frequency responses of both partially and fully fine-tuned DNNs to understand the effect of partial fine-tuning. Our experiments include two models: DRN-D-38 with 0.2% of weights fine-tuned (in Fig. 6) and Mask R-CNN with 0.8% of weights fine-tuned (in Fig. 7). By comparing Fig. 6b and 6c, we observe that fine-tuning the entire DRN and fine-tuning only 0.2% of the DRN weights with the same distorted dataset yield DNNs with close frequency responses – Both avoid looking at the noisy high-frequency components, compared to the original one in Fig. 6a. We have a similar observation from Fig. 7.

3.3. GearNN workflow

Based on the algorithm in §3.2, we design GearNN, a robust inference architecture that adapts only a small portion of the DNN, following the instantaneous input distortion.

Splitting the DNN (offline). The first step is to split the DNN into two parts – the *adaptor* to be changed in real-time following the input distortion level and the *base* that remains constant. The splitting follows §3.2.2: For each distortion level q, GearNN fine-tunes the entire DNN using distorted training data and compares the result with the original DNN, to obtain the distortion-sensitivity metric v_i^q of every DNN layer. Then, by solving the optimization problem in Eq. (2), GearNN picks a subset of layers to fine-tune, which is the *adaptor* $\mathcal{A}_q$ for distortion level q.

Partial fine-tuning (offline). GearNN then fine-tunes the *adaptors*, each using a dataset distorted to a particular

Algorithm 1 DNN splitting & partial fine-tuning (Offline).

Input: $\mathcal{Q} = \{q_1, \ldots, q_N\}$ – Supported distortion levels
$\mathcal{D}$ – Pre-trained DNN of K layers, each having size N_k
M – Size constraint of each adaptor
Output: $\mathcal{A}^q$ – Fine-tuned subset of DNN layers (*adaptor*)

for all $q \in \mathcal{Q}$ **do**
  Fine-tune $\mathcal{D}_q^* = f_q(\mathcal{D})$ with the data of distortion level q
  Compute $v_i^q = \frac{1}{N_i} \sum_{j=1}^{N_i} \|p_i^j - f_q(p_i^j)\|$ for each $\boldsymbol{l}_i \in \mathcal{D}$
  Obtain "values" list $\boldsymbol{v}^q = \{v_1^q, \ldots, v_K^q\}$ using Eq. (1)
  Obtain "costs" list $\boldsymbol{n} = \{N_1, \ldots, N_K\}$
  $S = knapsack(value = \boldsymbol{v}^q, cost = \boldsymbol{n}, bound = M)$
  The subset of layers to fine-tune is $\mathcal{A}_q = \{\boldsymbol{l}_i : i \in S\}$
  Fine-tune $\mathcal{A}_q^* = f_q(\mathcal{A}_q)$ with data of distortion level q

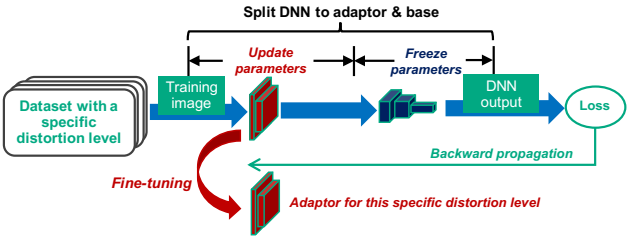


Figure 8: Partial weight fine-tuning workflow.

level (the original dataset before distortion is the same as the one used in the DNN splitting step), while freezing the *base*, as illustrated in Fig. 8. In this way, GearNN obtains multiple fine-tuned *adaptors* $\mathcal{A}_q^*$, each fitting a particular input distortion level. Alg. 1 summarizes the above steps.

Partial adaptation (online) With the fine-tuned *adaptors* and the *base*, we can run GearNN online. Since the size of adaptors is tiny, an inference server can load multiple adaptors for all supported distorted levels at low memory cost. Next, given a visual input stream with various distortion levels (*e.g.*, ABR streaming or receiving from heterogeneous IoT hardware), GearNN switches between the adaptors to fit the instantaneous input distortion level, while keeping the base weights unchanged, as shown in Fig. 1. It is straightforward to get the distortion level of the instantaneous input frame. For JPEG images, the quality information is embedded in the image; For DASH video streaming, the frame resolution is directly available; For the brightness levels, we can compute the brightness of an image based on its pixel values. Overall, GearNN can quickly determine which adaptor to use for the current input frame, enabling the real-time adaptation.

Why adapting at the layer-level? Adapting a subset of DNN weights requires not only their new values but also their positions in the model. For example, we can use a 0-1 vector to mark every weight (1 means do adaption and 0 means otherwise), then a DNN with millions of weights needs an extremely long vector, causing huge overhead. In contrast, labeling the DNN layers incurs tiny overhead since DNNs typically have up to hundreds of layers.

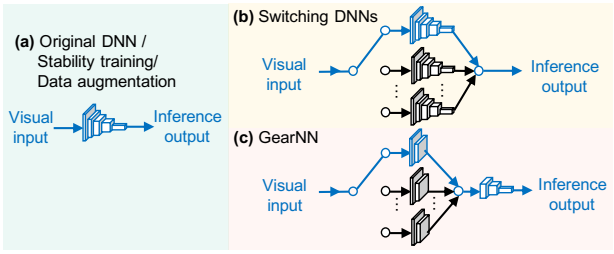


Figure 9: Implementation of GearNN and benchmarks. Modules in blue are active when processing an input instance, while modules in black are overhead.

4. Implementation

4.1. DNN-related configurations

We implement GearNN using PyTorch [14]. Since GearNN is designed as a generic inference architecture, we use two popular DNN models for its implementation: the dilated residual networks (DRN) [21, 22] and the Mask R-CNN model [7]. Also, we run DRN on the Cityscapes dataset [2] for urban driving scenery semantic segmentation and Mask R-CNN on the Penn-Fudan dataset [17] for pedestrian segmentation and detection. Our implementation uses a small learning rate of 0.001 and runs only 4 epochs when fine-tuning the adaptor as it only slightly updates the original DNN trained with the undistorted dataset.

Since GearNN allows a user to specify memory constraint, we evaluated GearNN with various adaptor sizes, ranging from 0.1% to 1% of the original DNN size, and our empirical results suggest that the adaptor with 0.2% of the original DRN size or 0.4% of the Mask R-CNN size best balance the inference accuracy and memory consumption.

4.2. Benchmark solutions

For benchmarks, we use 4 alternatives (also in Fig. 9):

Original DNN. The original DNN is a DNN pre-trained with the original undistorted dataset and is available in public. In our implementation, we download and use original DNN models from public links provided by their authors.

DNN switching. A straightforward solution to accommodate DNN inputs with various distortions is to train and use multiple DNNs, each for a particular distortion level.

Mixed training. A common technique to make DNN robust is to use perturbed images, via data augmentation, for training. We transform the original datasets to multiple versions, each for a particular distortion level, and then randomly sample from these distorted datasets to form a new dataset with mixed distortions. Finally, we fine-tune the DNN with the mixed dataset.

Stability training. Google proposed stability training [23] to improve the robustness of a DNN against noisy inputs. This solution forces a DNN to map inputs with small perturbations to their original input images to achieve the same inference output. *Stability training* only improves the training phase, and the inference still uses a constant DNN.

4.3. Distortion types

We now define input distortion types used in GearNN.

Resolution scaling. One primary motivation of GearNN is to make video analytics fit adaptive bitrate (ABR) video streaming, and the standard way of ABR streaming is to adapt the video resolution like MPEG-DASH [15]. Downscaling the video dimension reduces its size and causes distortions. We use H.264 encoder to compress the Cityscapes dataset of original resolution $\{2048 \times 1024\}$ to 6 different versions with smaller resolutions: $\{1920 \times 960, 1600 \times 800, 1280 \times 640, 1024 \times 512, 800 \times 400, 512 \times 256\}$. All these frames share a constant rate factor (CRF) of 18, meaning small quantization loss.

JPEG compression. The quality of a JPEG image is defined by an integer value Q, ranging from 1 to 100, and a higher Q indicates higher image quality and less distortion. We compress a lossless PNG-formatted dataset (like Cityscapes) to 10 different versions with JPEG quality levels, ranging from 10 to 100 in a step of 10, while keeping the same resolution as the undistorted dataset.

Brightness. A too bright or too dark visual input can significantly affect the DNN inference performance. We adjust the image brightness by Pillow to make multiple versions of the same dataset, each having a particular relative brightness to the original dataset. Our implementation includes relative brightness levels from 0.1 (10% of the original brightness) to 2.0 (double the original brightness), with the resolution unchanged.

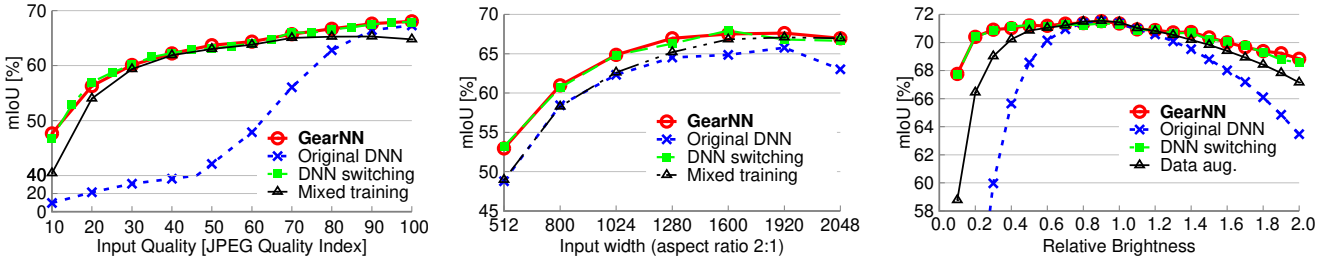
GearNN tackles distorted inputs by fine-tuning adaptors for each resolution, compression level, brightness, or even combinations of them. In §5, we evaluate different distortion types separately to show the impact of each type.

5. Experiments

We now present experimental results under different distortion types, datasets, tasks, and benchmarks.

5.1. JPEG quality level

In practical scenarios where a server (at edge or cloud) processes images from remote IoT devices, the inference engine at the server may encounter various quality levels of JPEG images as JPEG is the dominant image compression standard. In this context, we evaluate GearNN with inputs of various JPEG quality levels. We use the Cityscapes dataset, and build GearNN over DRN-D-22, a smaller DRN model than the one in §5.2, to show it is not limited to a particular model. Following §4.3, we convert the dataset to 10 versions with different JPEG quality levels and set the adaptor size as 0.2% of the model size (§4.1), then the memory overhead is $0.2\% \times (10-1) = 1.8\%$ over the *original DNN*.



(a) Different JPEG qualities. (b) Different resolutions. (c) Different brightness.

Figure 10: GearNN can accommodate various input distortion levels.

Table 1: Memory overhead.

	10 of JPEG qualities (DRN-D-22)		*7 of H.264 resolutions* (DRN-D-38)	
	Params	*Overhead vs. original*	*Params*	*Overhead vs. original*
GearNN	16,193,465	1.8%	26,334,993	1.2%
Original DNN	15,907,139	0%	26,022,723	0%
Stability training	15,907,139	0%	26,022,723	0%
Mixed training	15,907,139	0%	26,022,723	0%
DNN switching	159,071,390	900%	182,159,061	600%

The result in Fig. 10a confirms the high inference accuracy of GearNN on all JPEG quality levels. In contrast, *original DNN* suffers from significant accuracy degradation at low quality, and its mIoU falls below 40% for quality under 50. *Mixed training* also achieves lower accuracy than GearNN does, especially on inputs with high & low qualities, because it attempts to map different distorted versions of an image to the same inference output, which is difficult for a single DNN (Fig. 2). *DNN switching* yields accuracy similar to GearNN's, but at the cost of much higher memory cost (Table 1), as it needs to keep multiple DNNs in the memory. When each quality level has the same probability of appearing in the input, GearNN achieves 18.12% average accuracy gain over the *original DNN* and 1.95% gain over the *mixed training*. The accuracy difference with the memory-hungry *DNN switching* is only 0.22%.

5.2. H.264 video resolution

We then evaluate how GearNN guarantees high inference accuracy across dynamic input video resolutions, by using the Cityscapes dataset, DRN-D-38 model, and the frame scaling with 7 resolution levels as distortions (§4.3) We set the adaptor size to 0.2% of the model size (§4.1), then the memory overhead is $0.2\% \times (7-1) = 1.2\%$ over the *original DNN*. From the results in Fig. 10b, we see that both GearNN and *DNN switching* outperform the *original DNN* and *mixed training* significantly, with around 4% higher accuracy (mIoU) under resolutions lower than 1024[1].

Despite the similar accuracy of GearNN and *DNN switching*, GearNN only costs a 1.2% memory overhead over the *original DNN* (Table 1), while *DNN switching* requires keeping multiple DNNs in the memory and thus suffers from a high memory overhead. When each input resolution has an equal chance to appear, GearNN achieves 2.88% higher average accuracy over the *original DNN*, 1.70% over the *mixed training*, and 0.71% over *DNN switching*.

5.3. Brightness level

Another application scenario of GearNN is autonomous driving, where the camera can capture video frames with diverse brightness levels. In such a scenario, abrupt brightness change may cause even human eyes to malfunction temporarily. In this experiment, we reveal that the DNN-based computer vision also suffers from the same problem, and GearNN can address this by partially adapting the DNN to the brightness changes. Following §4.3, we alter the brightness of the datasets and use the DNR-D-38 model.

From Fig. 10c, we see the same performance ranking of tested solutions under various brightness levels – GearNN and *DNN switching* achieve comparable accuracy, while *mixed training* suffers significant accuracy loss on high or low input brightness. Next, the *original DNN* has the worst performance on all brightness levels. Overall, considering the 20 brightness levels from 0.1 to 2.0, GearNN costs merely $0.2\% \times 20 = 4\%$ memory overhead, compared to the original DNN, while *DNN switching* costs 1900% extra memory, rendering it impractical. If all brightness levels have the same chance of appearing in the input, GearNN achieves 5.11% higher average accuracy than the *original DNN*, 1.11% than the *mixed training*, and a slight 0.05% accuracy difference from *DNN switching*.

5.4. Different datasets and tasks

To demonstrate that GearNN is not bounded to a particular model or dataset, we run GearNN on pedestrian detection and segmentation with the Penn-Fudan dataset. We use

[1]The DNN trained with original images achieves the highest accuracy at a smaller input size (Fig. 10b), which implies that scaling the object to match field-of-view of the DNN kernel may compensate for the information loss. Nevertheless, this observation is out of the scope of this paper.

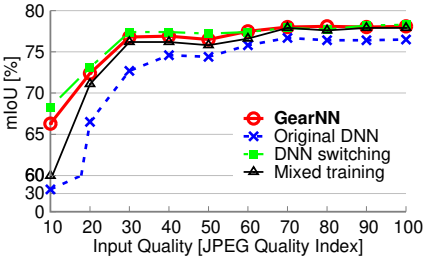


Figure 11: GearNN based on Mask R-CNN (semantic segmentation).

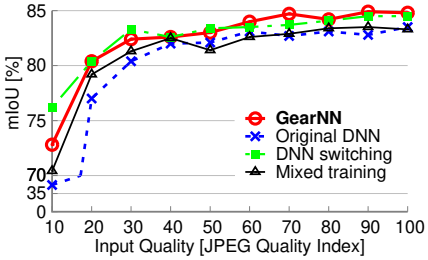


Figure 12: GearNN based on Mask R-CNN (bounding box detection)

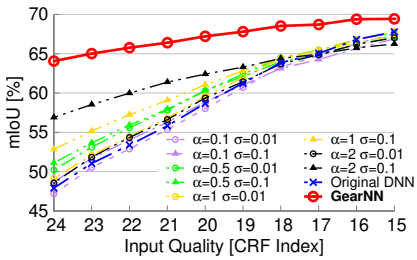


Figure 13: GearNN vs. *stability training* with various configurations.

Mask R-CNN with ResNet-50 as the backbone and perform both the segmentation and bounding box detection of the pedestrians. Our GearNN prototype runs on top of the Mask R-CNN model and supports 10 JPEG input quality levels (§4.3), for both pedestrian segmentation (Fig. 11) and bounding box detection (Fig. 12). Since we set the adaptor size to 0.4% of the model size (§4.1), the memory overhead is $0.4\% \times (10-1) = 3.6\%$ over the *original DNN*.

From the segmentation results in Fig. 11, we first see that the *original DNN* and *mixed training* suffer from severe accuracy loss when the input quality is below 20. GearNN has similar accuracy to *DNN switching*, as in all other experiments. As the input quality increases, the accuracy of the *original DNN* and *mixed training* remain below GearNN's. But the gap is smaller than that of the Cityscapes dataset because the PennFudan dataset has a lower resolution than Cityscapes, and semantic segmentation on the lower-resolution dataset is simpler and more robust to input distortions. In Fig. 12, we further plot the bounding box detection accuracy measured from the same set of experiments. The results show that GearNN always achieves higher accuracy than the *original DNN* and *mixed training*, especially at low input qualities. Meanwhile, it achieves similar accuracy, while maintaining much lower memory overhead than *DNN switching*.

Overall, if all JPEG quality levels have the same probability to appear in the input, GearNN achieves 5.16% (segmentation) / 3.53% (detection) higher average accuracy than the *original DNN*, 1.20% (segmentation) / 1.33% (detection) higher than the *mixed training*, and a small 0.43% (segmentation) / 0.24% (detection) accuracy difference from the memory-hungry *DNN switching*.

5.5. Comparison with stability training

There are only a few existing solutions in improving the DNN robustness to distorted/perturbed inputs. One of the most well-known works is the *stability training* proposed by Google. Therefore, we perform an extensive performance comparison against *stability training* under various configurations. In particular, we implement both GearNN and *stability training* based on the DRN-D-38 model and evaluate them on the Cityscapes dataset compressed by the H.264 encoder. For *stability training*, there are two configurable parameters: (i) α controls the tradeoff between the stability and the accuracy on undistorted inputs, and (ii) σ controls the tolerance to distortions. We test multiple combinations of α and σ, then show the results in Fig. 13.

We first observe that GearNN outperforms all tested combinations of α and σ at any input distortion level. We also see that α and σ control the performance of *stability training* as expected. Overall, a higher α or σ leads to better tolerance to distortions. It essentially sacrifices the performance on the low-distortion inputs to improve the performance on the high-distortion inputs, so that the overall performance is balanced. Compared to the *stability training* configuration with the highest accuracy when $\alpha = 2$ and $\sigma = 0.1$, GearNN achieves 4.84% average accuracy gain, while the highest accuracy gain is 7.15% at the CRF of 24 and 3.18% at the CRF of 15. In sum, *stability training* uses a single DNN to fit a wide range of distortions, and thus it has to tradeoff performance across different distortion levels. In contrast, GearNN partially adapts the weights to fit the current input distortion level, which guarantees its high accuracy and low memory overhead.

6. Conclusion

In this paper, we present GearNN, a memory-efficient adaptive DNN inference architecture, to combat the various DNN input distortions. GearNN enables robust inference under a broad range of input distortions, without compromising memory consumption. It identifies and adapts only the distortion-sensitive DNN parameters, a tiny portion of the DNN (*e.g.*, 0.2% of the total size), following the instantaneous input distortion level. Our evaluation demonstrates the superior performance of GearNN over benchmark solutions such as *stability training* from Google, when the distortion level of input varies due to adaptive video resolution, JPEG compression, or brightness change. As a general inference architecture, GearNN can be potentially applied to many existing DNN models and enables them to accommodate the various DNN input distortions in the IoT era.

References

[1] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for efficient inference. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2017. 2

[2] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 6

[3] Gunhan Dundar and Kenneth Rose. The effects of quantization on multilayer neural networks. *IEEE Transactions on Neural Networks*, 1995. 3

[4] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 2

[5] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 3

[6] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017. 2

[7] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2961–2969, 2017. 6

[8] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017. 2

[9] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *ACM SIGCOMM Computer Communication Review*. ACM, 2014. 3

[10] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 2017. 3

[11] Ehud D Karnin. A simple procedure for pruning back-propagation trained neural networks. *IEEE transactions on neural networks*, 1990. 2

[12] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 2009. 2

[13] Roger Pantos and William May. Http live streaming. Rfc, 2017. 3

[14] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017. 6

[15] Iraj Sodagar. The mpeg-dash standard for multimedia streaming over the internet. *IEEE Multimedia*, 2011. 3, 6

[16] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI Global, 2010. 2

[17] Liming Wang, Jianbo Shi, Gang Song, and I-fan Shen. Object detection combining recognition and segmentation. In *Asian Conference on Computer Vision*, pages 189–199. Springer, 2007. 6

[18] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018. 2

[19] Xiufeng Xie and Kyu-Han Kim. Source compression with bounded dnn perception loss for iot edge computer vision. In *The 25th Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2019. 3

[20] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, 2014. 2

[21] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *International Conference on Learning Representations (ICLR)*, 2016. 2, 6

[22] Fisher Yu, Vladlen Koltun, and Thomas Funkhouser. Dilated residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2, 6

[23] Stephan Zheng, Yang Song, Thomas Leung, and Ian Goodfellow. Improving the robustness of deep neural networks via stability training. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 1, 2, 6