Real-time Neural Network Inference on Extremely Weak Devices: Agile Offloading with Explainable AI

Kai Huang
University of Pittsburgh
USA
k.huang@pitt.edu

Wei Gao
University of Pittsburgh
USA
weigao@pitt.edu

ABSTRACT

With the wide adoption of AI applications, there is a pressing need of enabling real-time neural network (NN) inference on small embedded devices, but deploying NNs and achieving high performance of NN inference on these small devices is challenging due to their extremely weak capabilities. Although NN partitioning and offloading can contribute to such deployment, they are incapable of minimizing the local costs at embedded devices. Instead, we suggest to address this challenge via agile NN offloading, which migrates the required computations in NN offloading from online inference to offline learning. In this paper, we present *AgileNN*, a new NN offloading technique that achieves real-time NN inference on weak embedded devices by leveraging eXplainable AI techniques, so as to explicitly enforce feature sparsity during the training phase and minimize the online computation and communication costs. Experiment results show that AgileNN's inference latency is >6× lower than the existing schemes, ensuring that sensory data on embedded devices can be timely consumed. It also reduces the local device's resource consumption by >8×, without impairing the inference accuracy.

CCS CONCEPTS

• **Computer systems organization** → **Embedded and cyber-physical systems**; • **Computing methodologies** → **Artificial intelligence**;

KEYWORDS

Neural Network Inference, Microcontrollers, Offloading, Explainable AI.

ACM Reference Format:
Kai Huang and Wei Gao. 2022. Real-time Neural Network Inference on Extremely Weak Devices: Agile Offloading with Explainable AI. In *The 28th Annual International Conference On Mobile Computing And Networking (ACM MobiCom '22), October 17–21, 2022, Sydney, NSW, Australia.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3495243.3560551

ACM MobiCom '22, October 17–21, 2022, Sydney, NSW, Australia

ACM ISBN 978-1-4503-9181-8/22/10…$15.00
https://doi.org/10.1145/3495243.3560551

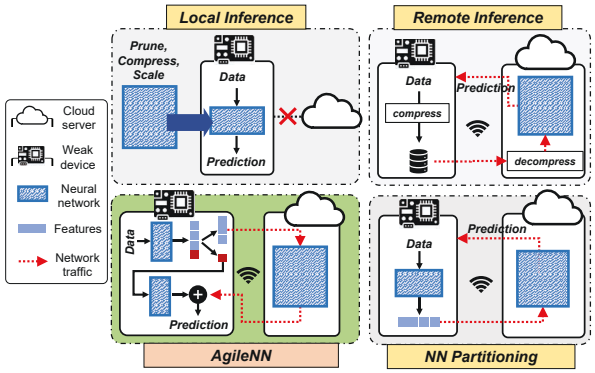


Figure 1: Existing work vs. AgileNN

1 INTRODUCTION

Neural networks (NNs) have been used to enable many new applications, such as face and speech recognition [6, 32], object tracking [5, 15], and personal assistants for business [66] and health [69]. With the penetration of these applications into our daily life, there is a pressing need of enabling real-time NN inference on small embedded devices, to allow more intelligent and prompt decision making on these weak devices. For example, on-device data processing on home security sensors [70] and industry actuators [1] will allow prompt response to sporadic events, and real-time analysis of human activity data on wearables could timely identify potential health risks [3, 13]. Deployment of NN models on small drones and robots is the technical foundation of these devices' autonomous navigation [7, 23], which is useful in many environment surveillance, disaster rescue and military scenarios. Furthermore, real-time NN inference, if made possible on energy-harvesting-powered sensors [24, 33] and RF-powered devices [35, 47], could expand the current horizon of AI to another magnitude.

Deploying NNs on these small devices, however, is very challenging due to the disparity between these devices' weak capabilities and NNs' high computing demands. For example, the ResNet50 model contains 23 million parameters and 50 convolutional layers [28], and requires at least 100MB memory and a processor of >2GHz to achieve 60ms inference latency on a smartphone [52]. Such amount of computing resources, however, is >10 times higher than what is available on a STM32 microcontroller (MCU)[1].

To eliminate such disparity, researchers aimed to reduce the NN complexity via compression [18, 25] or pruning [27, 51] (Figure 1 - top left), which remove redundant NN weights and structures.

[1]The STM32 MCUs have been widely used on embedded sensors and actuators. The STM32F746 MCU, for example, is equipped with an ARM Cortex-M7 processor running at 216 MHz and 320KB of local memory [2].

		Local comput. complexity	Memory cost	Data trans. cost	Inference accu. loss	Training cost
Local Inference	Compression [18, 25]	Very High	High	None	High	High
	Pruning [27, 51]	Very High	High	None	High	High
	NAS [10, 44]	High	Medium	None	Medium	Very High
Remote Inference	JPEG [62], MPEG [41]	Low	Low	High	Low	Low
	NN-favorable compression [45, 46]	Medium	Low	Medium	Medium	Medium
NN Partitioning [31, 34, 36, 39, 42, 65]		High	Medium	Low	Low	Medium
AgileNN		**Very Low**	**Low**	**Very Low**	**Very Low**	**Medium**

Table 1: Comparison of the approaches to NN inference on weak devices

However, the existing schemes mainly target strong mobile devices (e.g., smartphones) where a moderate reduction of NN complexity is sufficient. When being tailored to the weak embedded devices' extreme resource constraints, the over-simplified NNs will suffer large reductions of inference accuracy. For example, when the size of a ResNet50 model is reduced by 100 times, its inference accuracy could drop from 77% to 62% [22]. Even with the recent Neural Architecture Search (NAS) technique that finds the best NN structure with the given complexity constraint [10, 44], the inference accuracy loss could be still >10%.

Instead, a better solution to avoiding the inference accuracy loss is to offload the NN computations to a cloud server. To minimize the communication cost of offloading, one can compress the NN input data [41, 45, 46, 62] before transmission (Figure 1 - top right), but the compression ratio could be limited and result in high data transmission latency, with the low-speed wireless radios (e.g., Bluetooth and ZigBee) used on embedded devices for energy saving purposes. Later research efforts suggest to partition the NN (Figure 1 - bottom right), and use the *Local NN*[2] to transform the input data into a more compressible form of feature representations before transmission. Existing NN partitioning schemes [31, 34, 36, 39, 42, 65], however, need to use an expensive Local NN to enforce feature sparsity and incur unacceptable computing latency on the local device. The key reason of this limitation is that these schemes regardlessly apply the same learning approach to every input data, and hence need a sufficient amount of representation power at the local NN for the worst case of input data.

To address this limitation and practically enable NN inference on extremely weak devices (e.g., MCUs) with the minimum latency, in this paper we present *AgileNN*, a new technique that shifts the rationale of NN partitioning and offloading from fixed to agile and data-centric. Our basic idea is to incorporate the knowledge about different input data's heterogeneity in training, so that the required computations to enforce feature sparsity are migrated from online inference to offline training. More specifically, we interpret such heterogeneity as different data features' importance to NN inference, and leverage the eXplainable AI (XAI) techniques [56, 59] to explicitly evaluate such importance during training. In this way, as shown in Figure 1 - bottom left, the online inference can enforce feature sparsity by only compressing and transmitting the less important features, without involving expensive NN computations. The important features, on the other hand, are retained at the local device and can be perceived by a lightweight NN with low complexity. Predictions from Local NN and Remote NN, eventually, are combined at the local device for inference.

The major challenge of using AgileNN in practice, however, is that different data features may have similar importances to NN inference. In this case, sparsity among less important features will be reduced and result in lower data compressibility, and more features also need to be retained at the local device, incurring extra computing latency. To address this challenge and simultaneously minimize the local embedded device's costs in computation and communication, AgileNN's basic approach is to intentionally manipulate the data features' importance via non-linear transformation in the high-dimensional feature space, so as to ensure that such importance's distribution over different features is skewed. In other words, only few features make the majority of contributions to NN inference. In our design, we realize such skewness manipulation with a highly lightweight feature extractor, and jointly train the feature extractor with Local and Remote NNs to ensure inference accuracy.

To our best knowledge, AgileNN is the first technique that achieves real-time NN inference on embedded devices with extremely weak capabilities in computation and communication. Our detailed contributions are as follows:

- We effectively migrate the required computations in NN offloading from online inference to offline training, by leveraging XAI techniques that allow lightweight enforcement of feature sparsity at runtime.
- We developed new AI techniques that use XAI to explicitly manipulate the importances of different data features in NN inference, so as to ensure the effectiveness of NN partitioning and offloading.
- By enforcing skewness of such importance's distribution over different features, we allow flexible tradeoffs between the accuracy and cost of NN inference on embedded devices, without incurring any extra computing or storage cost.

We implemented AgileNN on a STM32F746 MCU board and a server with an Nvidia RTX A6000 GPU, and evaluated the performance of AgileNN on various popular datasets under different system conditions. From our experiment results, we have the following conclusions:

- AgileNN is *real-time*. Compared to the existing schemes [39, 44, 65], AgileNN reduces the NN inference latency by up to 6x, and restrains such latency within 20ms on most datasets. It hence supports real-time NN inference on weak

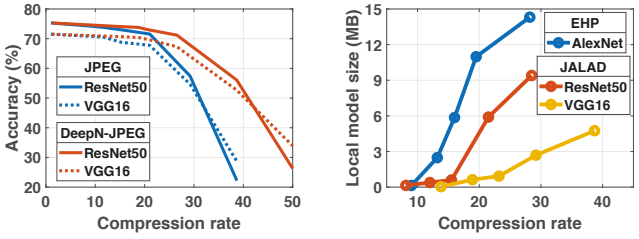
[2]In this rest of this paper, we use *Local NN* to indicate the portion of partitioned NN at the local device, and *Remote NN* to indicate the portion of partitioned NN at the cloud server.

embedded devices, by ensuring that the sensory data can always be timely consumed.

- AgileNN is *accurate*. Compared to the existing NN partitioning schemes, AgileNN provides the similar inference accuracy but achieves much higher feature sparsity. Such high sparsity, then, reduces the amount of data transmission in NN offloading by up to 70%.
- AgileNN is *lightweight*. Compared to the current NN inference schemes on embedded devices, AgileNN reduces the local energy consumption by >8x, while consuming 1.2x less memory and 5x less storage space.
- AgileNN is *adaptive*. It minimizes the performance degradation of NN inference in different embedded device settings and system conditions, even with extremely low computing power and wireless bandwidth.

2 BACKGROUND & MOTIVATION

To help better understand the AgileNN design, we first demonstrate the limitations of the existing NN offloading schemes. Then, we motivate our design by introducing XAI techniques that explicitly evaluate the importance of different features, and highlighting the necessity of having features with skewed importance distributions.



(a) Compressing the raw input data (b) Feature extraction

Figure 2: Data compressibility in NN offloading

2.1 Data Compressibility in NN Offloading

To reduce the communication cost of NN offloading, an intuitive method is to compress the raw data before transmission, but heavy compression will distort the important information in data and hence affect the NN inference accuracy. To verify such impact, we apply both standard JPEG [62] and NN-favorable DeepN-JPEG [45] compression methods to images in the ImageNet dataset [17], and measure the inference accuracy loss on various NN models when using different data compression rates [28, 55, 58]. As shown in Figure 2(a), a moderate compression rate of 25x will reduce the NN inference accuracy by >10%, and such accuracy loss will quickly grow to >20% when the compression rate is >30x.

Instead, current NN partitioning approaches improve the data compressibility by extracting more compressible forms of feature representations from the raw input data. However, as shown in Figure 2(b) with two representative partitioning approaches (EHP [36] and JALAD [42]), although these schemes can achieve the similar compression rates with the minimum impact on the NN inference accuracy[3], their feature extraction is very computationally expensive. For example, achieving a compression rate of 30x will require a large Local NN with a model size of >3MB, which is unaffordable on most weak embedded devices such as STM32 MCUs.

2.2 Explainable AI

The aforementioned limitation motivates our design that achieves better data compressibility by evaluating different data features' importance during offline training. Based on such knowledge about feature importance, during online inference we can explicitly enforce sparsity in the less important features with the minimum local computing cost. To evaluate such feature importance, classic perturbation-based approaches [12] measure how the NN inference accuracy varies after injecting noise to features in all the training data, but cannot precisely evaluate feature importance over individual data samples. Attention-based mechanisms [8, 61] support such individualized evaluation by adding an extra weight generator in NN training, but need to tailor the weight generator's structure to each NN model and could hence be inaccurate in some NN models.

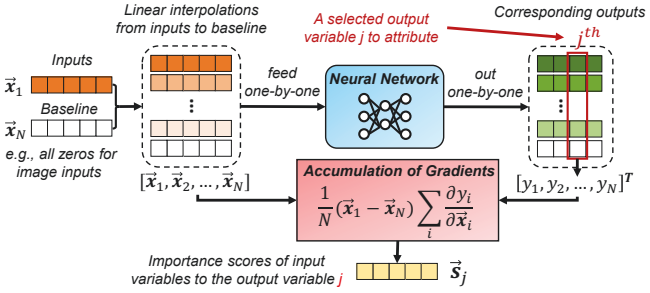
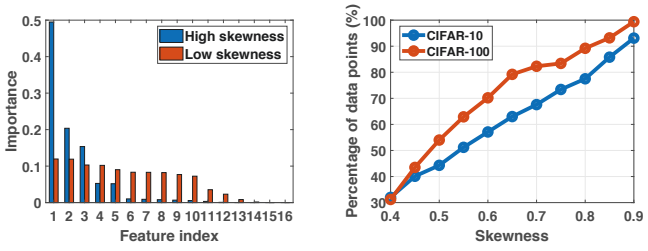


Figure 3: Integrated Gradients

Recent research on eXplainable AI (XAI) improves the accuracy of feature importance evaluation by offering attribution tools that quantitatively correlate each input variable to the NN outputs during training [56, 59]. For example, typical XAI tools such as Integrated Gradients (IG) [59], as shown in Figure 3, feed a number of linear interpolations between the input variables and a naive baseline to the NN. Then, for an input variable, they compute each of its interpolation's gradient with respect to the NN's output (e.g., confidence scores), and accumulate these gradients to measure the importance of this input variable[4]. In this way, these XAI tools are robust and applicable to any AI model without extra modification.



(a) Different levels of skewness (b) Skewness CDF

Figure 4: Skewness of feature importance. Skewness is measured as the normalized importance of the top 20% features, using the MobileNetV2 model [55].

[3]The loss of NN inference accuracy in these schemes can be effectively restrained within 1%, for all the data compression rates being applied.

[4]In practice, such accumulation is used to approximate to the path integral of gradients. The more interpolations are used, the better approximation will be.

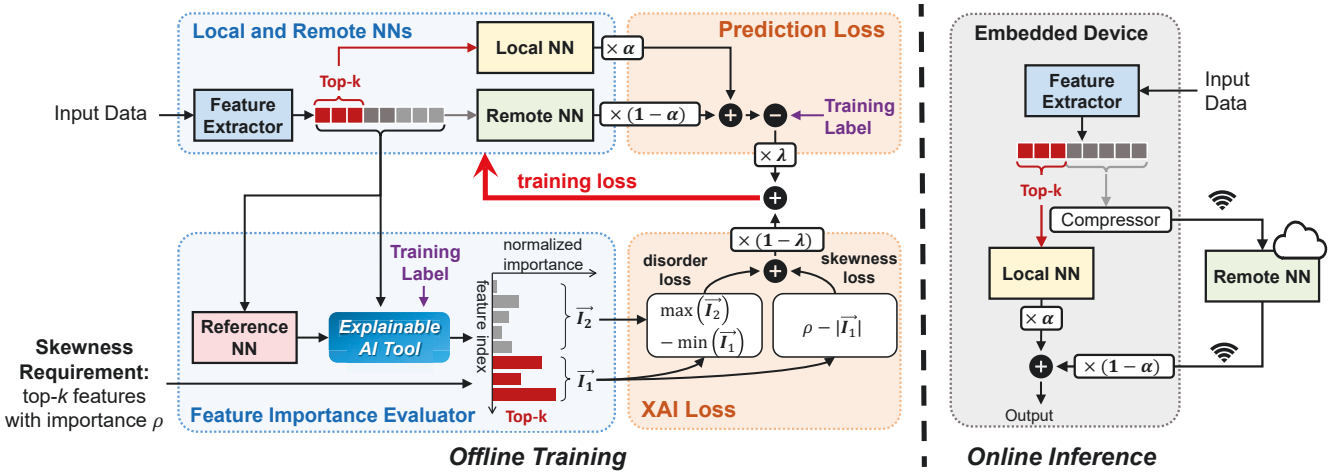


Figure 5: Overview of AgileNN design

However, one key limitation of the existing XAI tools is that its accuracy of feature importance evaluation builds on accurate NN inference in advance. If the NN's output is ambiguous (e.g., due to inadequate training), XAI could produce misleading evaluations because the gradients computed from the NN's output are highly random. In the worst case, such randomness can cause all the features to be misranked by their importance [54]. This limitation motivates us to use a pre-trained reference NN model in AgileNN's training, to ensure correct XAI evaluation on feature importance.

2.3 Skewness of Feature Importance

Based on the feature importance evaluated by XAI, the effectiveness of AgileNN's offloading depends on the skewness of such importance's distribution over different features. The higher such skewness is, the fewer features are playing a dominant role in NN inference and we can hence enforce higher sparsity in less important features without impairing the NN inference accuracy. However, as shown in Figure 4(a) that exemplifies such importance distribution of different data samples in the CIFAR-10 dataset [38], skewness may not always exist in every input data. Furthermore, as shown in Figure 4(b), when we measure skewness as the ratio of normalized importance of the top 20% features, such skewness in >40% of data samples in the CIFAR-10 and CIFAR-100 datasets [38] is <50%.

Such low skewness in the input data motivates us to design new NN structures that intentionally manipulate and enhance such skewness in feature extraction, while minimizing the impact of such skewness manipulation on the NN inference accuracy.

3 SYSTEM OVERVIEW

As shown in Figure 5, AgileNN partitions the neural network into a Local NN and a Remote NN. In online inference, AgileNN runtime uses a lightweight feature extractor at the local embedded device to provide feature inputs: the top-k features with high importance are retained by the Local NN to make a local prediction, which is then combined with the Remote NN's prediction from other less important features for the final inference output. In this way, the complexity of Local NN could be minimized without impairing the inference accuracy, and high sparsity can be enforced when compressing and transmitting less important features to the server.

In offline training, AgileNN jointly trains the feature extractor, Local NN and Remote NN with a unified loss function. In particular, the feature extractor is trained to meet the user's *requirement of feature importance skewness*, such that the normalized importance of top-k features should exceed a threshold $\rho \in [0, 1]$. During the training process, enforcing this requirement is equivalent to apply non-linear transformations to the output feature vector in the high-dimensional feature space.

Based on this design, AgileNN can flexibly balance between the accuracy and cost of NN inference by adjusting the required feature importance skewness. The higher the skewness is (i.e., smaller k and larger ρ), the lower resource consumption will be at the local device due to the higher compressibility of less important features being transmitted, but the NN inference is more affected due to the feature extractor's non-linear transformation in the feature space. In practice, with the same AgileNN runtime being trained for the specific embedded device, the user can adaptively choose different tradeoffs according to the application scenarios and local resource conditions, without spending extra local computing or storage resources to maintain multiple NN models [21] or adopt different learning strategies [39].

3.1 Skewness Manipulation

In order to manipulate the importance of extracted features and meet the skewness requirement, AgileNN's basic approach is to incorporate both the inference accuracy and current skewness of feature importance into the unified loss function in training. More specifically, in each training epoch, AgileNN feeds the current set of features extracted by the feature extractor to the XAI tool module, which evaluates and outputs the importance of each feature to NN inference. The skewness of feature importance, then, is incorporated into the loss function in the following two aspects.

1) The *disorder loss*, which mandates that the top-k features with highest importance are always in the first k channels of the output feature vector. It is calculated as

$$L_{\text{disorder}} = \max\left(0, \max(\vec{I}_2) - \min(\vec{I}_1)\right), \tag{1}$$

where $\vec{I}_1$ indicates the normalized importances of features in the first k channels of the output feature vector and $\vec{I}_2$ indicates the normalized importances of other features.

This ordering is essential to online inference, where the XAI tool is unavailable and the top-k features with high importance should hence be always located in fixed channels of the extracted feature vector. With such feature ordering, we can further instruct the feature extractor to enhance the importance of the top-k features and hence enforce the required skewness. The knowledge about the fixed locations of top-k features in the feature vector, on the other hand, will also enable prompt split of features for local and remote inferences, without involving any extra computations or manual efforts at run-time. More details of such feature ordering are provided in Section 4.1.

2) The *skewness loss*, which measures the difference between the current skewness of feature importance and the skewness requirement. It is calculated as

$$L_{\text{skewness}} = \max\left(0, \rho - |\vec{I}_1|\right), \tag{2}$$

where $|\cdot|$ indicates the vector's 1-norm.

These two loss components are then combined with the standard prediction loss in AgileNN's training. Details about such combined training loss are provided in Section 4.2.

On the other hand, as described in Section 2.2, the accuracy of XAI's feature importance evaluation requires a well-trained NN in advance to provide correct inference labels. Hence, to ensure the quality of AgileNN's training, we introduce a reference NN model, which has been pre-trained for the same learning task with sufficient representation power[5], to provide inference outputs to the XAI tool using the extracted features from AgileNN's feature extractor. To further avoid any possible ambiguity, we compare each inference output made by the reference NN with the training label, and only use it in XAI evaluation if the reference NN makes correct predictions.

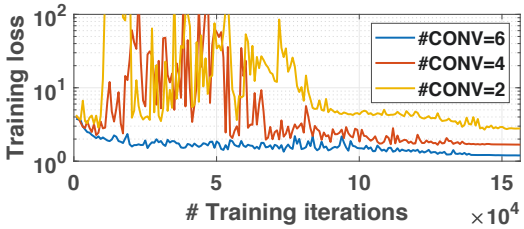


Figure 6: Training stability with different numbers of convolutional layers in feature extractor

3.2 Pre-processing the Feature Extractor

AgileNN jointly trains the Local NN and Remote NN with the feature extractor, so as to ensure that they can provide accurate predictions from the extracted features with skewed distribution of importance. However, since the feature extractor in AgileNN needs to be deployed at the local device and hence has to be very lightweight, it may not have sufficient representation power to meet this learning objective in the initial phase of training, and the joint training may hence encounter unexpectedly high learning difficulty or even fail to converge. For example, as shown in Figure 6, such joint training on CIFAR-100 dataset, if starting from scatch, is highly unstable unless a sufficient number of convolutional layers ($\geq$6) is used in the feature extractor.

To avoid such learning difficulty, AgileNN's approach is to pre-process the feature extractor and initialize its network weights, prior to the joint training with Local and Remote NNs. In this way, the joint training will not start from scratch but instead from a more established stage with less ambiguity, and hence has lower requirement on the initial representation power of the feature extractor.

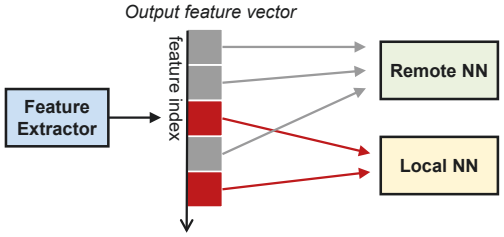


Figure 7: Pre-processing the feature extractor

More specifically, the feature ordering mandated by the disorder loss in Section 3.1 may be hard to fulfill by the feature extractor in the initial stage of training. Instead, as shown in Figure 7, we select k initial channels in the output feature vector where the top-k features with high importance are most likely to be located. We then use the corresponding k features as the input to the Local NN. More details of selecting these initial channels and integrating such pre-processing into the training process are in Section 5.

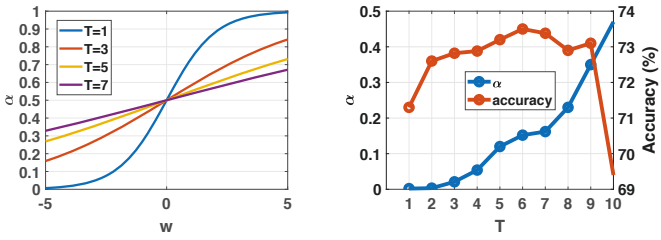
3.3 Combining Local and Remote Predictions

AgileNN combines the predictions made by the Local and Remote NNs via weighted summation, to produce the final inference output at the local embedded device. Compared to other NN-based alternatives such as adding an extra NN layer for combination, we use this solution because of the following two reasons. First, computing such point-to-point weighted sums is much more lightweight than NN operations and adds negligible computation overhead to the local device. Second, the outputs of Local and Remote NNs always correspond to the same number of aligned feature channels, and the point-to-point summation retains such alignment. Using an NN layer (e.g., a fully-connected or convolutional layer) to combine these two outputs, on the other hand, could possibly entangle them together and break such alignment, hence impairing the final inference accuracy.

The main difficulty of such combination, however, is that the outputs of Local NN and Remote NN may not be in the same scale and may hence result in extra loss in inference accuracy, because some small but important output values in one NN could be overwhelmed by large values in another NN. To address this difficulty, our solution is to incorporate the summation weight α into the joint training procedure. Being the same as other NN parameters, α is also trained with gradient-based feedback using stochastic gradient descent (SGD) algorithms [11]. However, due to the big difference between the complexities of Local NN and Remote NN, the training is likely to be biased towards the Remote NN and ignore the Local

[5]In practice, such reference models are widely available as public online. For example, EfficientNet model is available online [60] and can achieve >90% inference accuracy on large datasets such as ImageNet [17].

NN's contribution, by assigning near-zero values to α. Such bias could possibly make the training to be highly unstable or significantly reduce the inference accuracy, because the Local NN that perceives the top-k important features may not be well trained.



(a) Impact of T on α (b) Choices of T

Figure 8: Prediction weighting with α

To avoid this bias, in AgileNN we introduce a soft constraint by formulating α as a parameterized sigmoid function:

$$\alpha(w;T) = \frac{1}{1+e^{-w/T}},$$

where w is a trainable parameter and T controls α's sensitivity to w. As shown in Figure 8(a), the higher T is, the slower $\alpha(w;T)$ varies along with w, and hence the less likely that the value of α will approach 0 or 1 during training. In practice, as shown in Figure 8(b), a moderate value of T between 4 and 8 can effectively avoid biased values of α and ensure high inference accuracy.

The trained value of α is loaded to AgileNN runtime at the local device. In real-world settings, when the feature extractor does not correctly evaluate the importance of some features due to the possible inaccuracy in XAI, the user could flexibly fine-tune AgileNN's strategy of NN partitioning at run-time by reconfiguring the value of α, to mitigate the loss of inference accuracy.

4 SKEWNESS MANIPULATION

In this section, we provide technical details about how the training loss function in AgileNN's training is constructed based on the feature importances evaluated by XAI, so as to enforce the required skewness of such importances among the extracted features.

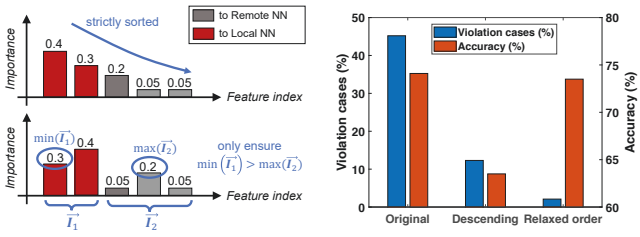
4.1 Feature Ordering

Since XAI evaluation of feature importance builds on accumulating gradients in training and is hence unavailable during online inference, AgileNN makes sure that its feature extractor always generates the top-k features with highest importance in the first k channels in the output feature vector, as shown in Figure 9(a) - top, so that AgileNN runtime at the local embedded device can correctly identify them for every input data during inference.

In training, a straightforward method to achieve this learning objective is to adopt the following loss function:

$$L_{\text{descent}} = \|\vec{I} - \vec{I}_{\text{sorted}}\|_2^2,$$

where $\vec{I}$ denotes the normalized importances of the currently extracted features and $\vec{I}_{\text{sorted}}$ denotes the sorted form of $\vec{I}$ in the descending order. Minimizing this loss, hence, ensures that the extracted features are always sorted in the descending order of their importances. However, strictly enforcing such descending order



(a) Different feature orders (b) Effectiveness of reordering

Figure 9: Feature reordering

in the produced feature vector requires high representation power in the feature extractor, or adds extra confusions in training if the feature extractor being used is too lightweight. To demonstrate this, we conduct preliminary experiments by using the feature extractor of the MobileNetV2 model [55] on the CIFAR-100 dataset [38]. As shown in Figure 9(b), enforcing such descending order in the output feature vector reduces the inference accuracy by >10%.

Instead, we relax the learning objective by reducing the number of features being repositioned. As shown in Figure 9(a) - bottom, we do not require that all the features are sorted in the descending order of their importance, but instead only require that any top-k feature's importance is higher than any other feature's importance. If any violation is found during training, a penalty will feedback to the NN for parameter update. Based on this relaxed learning objective, we construct our disorder loss as shown in Eq. (1), which will only be non-zero if any violation of feature ordering occurs in training. In theory, this loss function of feature disordering is almost always differentiable [48] and can be seamlessly incorporated into the regular training procedure[6]. As shown in Figure 9(b), L_{disorder} can reduce the percentage of disorder cases to <2% without impairing the inference accuracy.

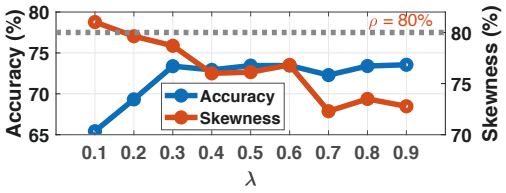


Figure 10: Impact of λ on CIFAR-100 dataset

4.2 Combined Training Loss

To enforce the skewness requirement, we want that the cumulative normalized importance of top-k features exceeds the given threshold ρ, and hence define the skewness loss as shown in Eq. (2). Then, we combine the disorder loss and skewness loss together to construct the training loss for skewness manipulation as:

$$L = \lambda \cdot L_{\text{prediction}} + (1-\lambda) \cdot (L_{\text{skewnss}} + L_{\text{disorder}})$$

where $L_{\text{prediction}}$ is the standard prediction loss and λ is a hyperparameter within $(0,1)$ to control the contributions of L_{skewnss} and L_{disorder} in training feedback. In practice, according to our preliminary results in Figure 10, aggressively reducing λ, although

[6]NNs are typically trained by providing gradient-based feedback being calculated from the loss function.

Algorithm 1 Selecting the k initial feature channels

Input: $\mathbf{D}_{\text{train}}$: the training dataset with N samples;
$\mathcal{T}_{XAI}(\cdot)$: XAI-enabled Feature Importance Evaluator;
$\mathcal{E}(\cdot)$: Feature extractor that outputs C channels
Output: $(j_1, j_2, ..., j_k)$: The k selected feature channels.

```
(p_1, p_2, ..., p_C) ← 0                                   //initialize
for each d_i ∈ D_train do
    F ← E(d_i)                                             // extract features
    I ← T_XAI(F)                                           //evaluate feature importance
    F_sorted ← sort_I(F)                                   //sort features by their importance in descending order
    F_top-k ← F_sorted[1 : k]                              //extract the top-k features with high importance
    for c = 1,...,C do
        if F[c] ∈ F_top-k then
            p_c ← p_c + 1/N
R ← argsort(p_1, p_2, ..., p_C)                            //get the ranking of channels by their likelihood
(j_1, j_2, ..., j_k) ← R[1 : k]                            //decide top-k channels
```

achieving higher skewness, could reduce the impact of prediction loss in training feedback and hence impair the NN inference accuracy. In contrast, we observe that a moderate value of λ between 0.2 and 0.4 could effectively approximate to the skewness requirement with the minimum impact on NN inference accuracy. Alternatively, one can also adopt the techniques on NN loss balancing [14, 26] for adaptive adjustment of λ at runtime.

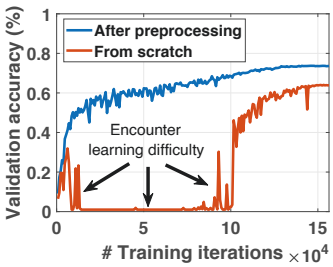


Figure 11: Effectiveness of Pre-processing

5 PRE-PROCESSING THE FEATURE EXTRACTOR

In this section, we describe in detail how we pre-process the feature extractor by selecting the initial k feature channels as the input to the Local NN in joint training. Intuitively, we can randomly select k feature channels and mandate the feature extractor to produce the top important features in these channels using the disorder loss described in Section 3.1. However, such arbitrary selection will bring serious learning difficulty that leads to low training quality. We demonstrate this by doing preliminary experiments on the CIFAR-100 dataset with such random channel selection. As shown in Figure 11, the NN experiences learning difficulty from the beginning epochs and it eventually causes poor convergence.

Instead, we make such channel selection based on the likelihood that one of top-k features with high importance is located in a channel, and compute such likelihood from the training data. More specifically, as described in Algorithm 1, such likelihood of each channel is cumulatively computed from all the N data samples in the training dataset, and increases by $1/N$ every time when a data sample's top-k features with high importance are located in the channel. As shown in Figure 11, our pre-processing can largely reduce the learning difficulty and ensure the quality of training.

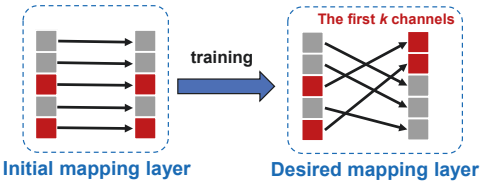


Figure 12: Training the mapping layer

After having selected these initial k channels, we expect the joint training process will be able to gradually enforce the required feature ordering, as described in Section 4.1, through the disorder loss. To facilitate this, as shown in Figure 12, in AgileNN's training we add an extra mapping layer between the feature extractor and the Local NN, and instruct the training process to ensure that the top-k important features reside in the first k channels of the output feature vector. After the training finishes, this mapping layer will be discarded and only the feature extractor is used in inference.

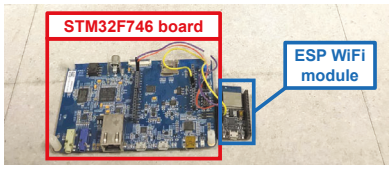


Figure 13: Devices in our implementation

6 IMPLEMENTATION

As shown in Figure 13, we use a STM32F746NG MCU board[7] as the local embedded device, which is widely used as the computing platform in current tinyML and on-device AI research (e.g., MCUNet [44]). It is equipped with an ARM 32-bit Cortex-M7 CPU at 216MHz, 320KB SRAM and 1MB flash storage, and supports flexible CPU frequency scaling to provide different amounts of on-device computing power. In addition, since neural network inference on the Cortex M series of MCUs has been officially supported by the TensorFlow community[8], we believe that using these MCUs to implement and evaluate AgileNN could better justify AgileNN's practical merits, compared to using other MCUs such as the MSP430 series.

The MCU board uses an ESP-WROOM-02D WiFi module to transmit data to a server. The server is a Dell Precision 7820 workstation that equips with a 3.6GHz 8-core Intel Xeon CPU, 128GB main memory and an Nvidia RTX A6000 GPU with 48GB memory.

As shown in Figure 14, our offline training in AgileNN is implemented using TensorFlow Python library, and we converted the Local NN from a `float32` model into an `int8` model using TensorFlow Lite Converter. This model is then casted to a static binary array for better memory efficiency on the local device. We use TF Micro runtime to execute the `int8` model on the STM32 board. To further reduce the Local NN's computing latency, we merge the

[7] https://www.st.com/en/microcontrollers-microprocessors/stm32f746ng.html
[8] https://www.tensorflow.org/lite/microcontrollers

original TF Micro runtime with CMSIS-NN 5.0, which provides extra acceleration on several selected NN operations on ARM devices. On the other hand, the Remote NN remains full precision and is executed by TensorFlow Python runtime on the server.

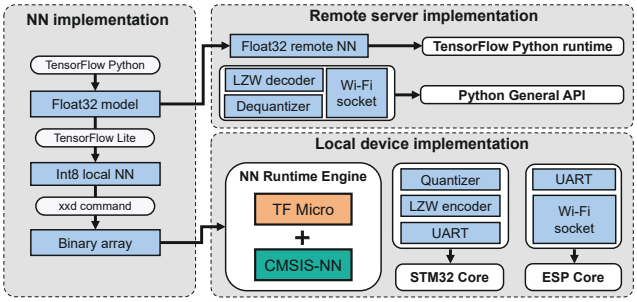


Figure 14: AgileNN implementation

On the STM32F746 board, we use STM32CubeIDE to implement its software in C++ and configure the embedded hardware. To compress the less important features before transmission, we first adopt learning-based quantization [4] and then apply standard LZW compression [49]. The compressed features are delivered to the WiFi module through UART, and the WiFi module transmits these features to the server through a UDP link at 6 Mbps.

On the server, we write a custom Python script to communicate with the STM32F746 board via general socket APIs, and verifies the integrity of the received features being applying them to the Remote NN.

7 PERFORMANCE EVALUATION

In our evaluations, to meet the embedded device's local resource constraints, we construct AgileNN's feature extractor with two convolutional layers, each of which has 24 output channels. The Local NN in AgileNN has the minimum complexity, and contains one global-average pooling layer and one dense layer. The Remote NN in AgileNN is constructed by removing the first convolutional layer from the MobileNetV2 [55] model. In all the evaluations, the sizes of feature extractor, Local NN and Remote NN in AgileNN remain fixed, but we vary the compression rate when transmitting the set of less important features from local to remote.

We evaluate AgileNN over multiple datasets listed below, and scaled all images in datasets to 96x96 in our experiments. Due to the low memory capacity of the embedded device, we focus on image recognition tasks instead of memory-demanding learning tasks, such as audio and video analytics [6, 64] that require expensive preprocessing steps [43].

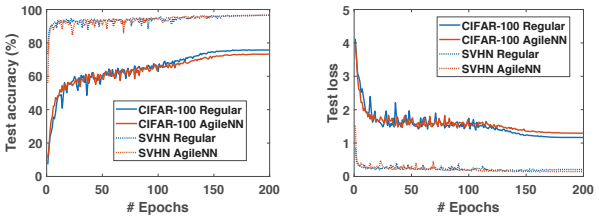
- **CIFAR-10/100 [38]**: This dataset contains 50k training images and 10k testing images that belong to 100 different categories and 10 super categories.
- **SVHN [50]**: This dataset contains 73k training images and 26k testing images about street address numbers.
- **ImageNet-200 [40]**: This is a subset of ImageNet dataset [17] that contains 100k training images and 10k testing images that are classified into 200 categories.

In training, AgileNN adopts an EfficientNetV2 CNN [60] that is pre-trained on the ImageNet dataset as the reference network, and the training hyperparameters are configured the same as MobileNetV2's setting [55]. We use the SGD optimizer with a learning rate of 0.1, and the standard weight decay is set to 5×10^{-4} and all the training runs for 200 epochs. The batch size in training is set to be 128 for the CIFAR-10 dataset and 64 for all other datasets.

In our evaluations, all the experiment results are averaged over the entire testing dataset. We compare AgileNN with the baseline of edge-only inference and three existing approach NN inference approaches, which span both categories of local inference and NN partitioning:

- **Edge-only inference**: The entire local data is compressed by the LZW compressor and transmitted to the server for inference.
- **MCUNet [44]**: The entire NN is running at the local embedded device, and the NN structure is optimally discovered by NAS according to the on-device resource constraint on the NN complexity.
- **DeepCOD [65]**: A NN-based encoder is embedded on the local device to transform the raw data or features into a more compressible form. The encoder is trained with the sparsity constraint in an end-to-end manner[9].
- **SPINN [39]**: Besides NN partitioning, early-exit structures are incorporated in the NN to adaptively adjust the NN complexity for runtime inference.

In particular, since MCUNet's NN design adopts different input resolutions for different datasets, we make sure to always use the same image resolution among all other approaches, including AgileNN, to make fair comparisons among different approaches.



(a) Test accuracy (b) Test loss

Figure 15: AgileNN's training performance on CIFAR-100 and SVHN datasets

7.1 Training Convergence and Cost

As a prerequisite, we first evaluate the quality and cost of AgileNN's training. As shown in Figure 15, during the training procedure, AgileNN exhibits a very similar rate of training convergence, in terms of test accuracy and loss, compared to regular training of MobileNetV2 on CIFAR-100 and SVHN datasets. These results show that, although the added feature ordering and skewness manipulation increases the learning complexity, AgileNN can still ensure fast training convergence with the appropriate loss function design and preprocessing of the feature extractor.

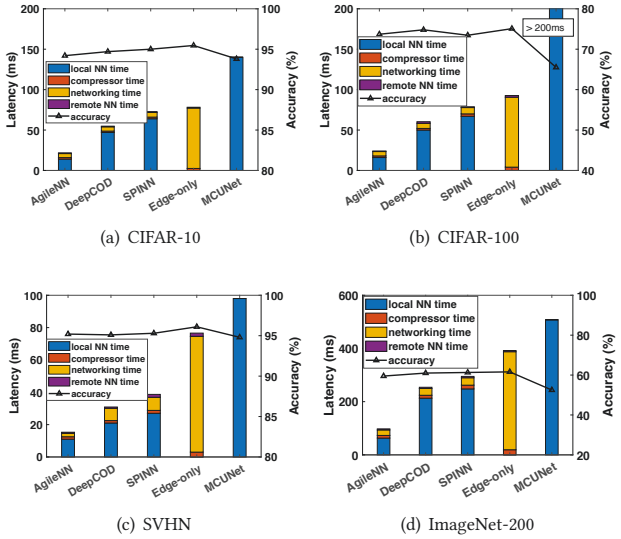
On the other hand, with the extra computations of feature importance using XAI and the corresponding involvement of extra

[9] AgileNN is equivalent to DeepCOD [65] if the top-k features with high importance are also compressed and sent to the server.

training feedback, we observe 3x-4x wall-clock time increase for each training epoch in AgileNN. However, since the training of feature extractor, Local and Remote NNs is conducted offline, such time increase will not affect AgileNN's online performance on weak embedded devices. Reduction of such training time can be done by either using stronger computing hardware (e.g., stronger GPUs) or more lightweight XAI tools [29, 57].

7.2 Accuracy and Latency of NN Inference

In general, the accuracy of NN inference can be improved by using more complicated NNs, which in turn result in longer inference latency. For easier comparisons, we configure the existing approaches' NN complexities so that the difference between their's and AgileNN's inference accuracy is always within 10%. In these cases, we compare the AgileNN's end-to-end inference latency with theirs. Note that for local inference approaches such as MCUNet, the inference latency is only determined by the local NN computing time. For NN partitioning approaches including DeepCOD, SPINN and AgileNN, the inference latency consists of 1) the local NN computing time, 2) the local computing time for data compression, 3) the network transmission time and 4) the remote NN time for data decompression and computing.



(a) CIFAR-10 (b) CIFAR-100

(c) SVHN (d) ImageNet-200

Figure 16: Latency and accuracy of NN inference

Results in Figure 16 show that, AgileNN is able to reduce the end-to-end inference latency by 2x-2.5x when compared to all the existing approaches, while retaining similar inference accuracy with DeepCOD and SPINN. In particular, such latency in most datasets can be effectively controlled within 20ms, which is comparable to the sampling interval of many embedded sensor devices[10]. As a result, AgileNN can effectively support real-time NN inference on weak embedded devices, by ensuring sure that generated sensory data can always be timely consumed.

[10]For example, most camera sensors on embedded devices have a sampling rate of 30Hz during video capture. The sampling rate of environmental sensors (e.g., temperature sensors) is usually <10Hz due to the slower changes of the physical environment [9]. The sampling rate of IMU sensors is usually capped at 100Hz, but a lower rate is used more often in practice to save power [19].

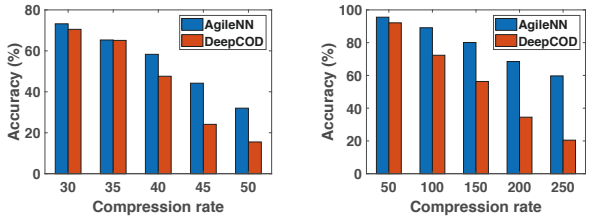
More specifically, Figure 16 shows that the AgileNN's latency reduction mainly comes from the lower local NN computing time, which can be reduced by up to 10x. Compared to DeepCOD and SPINN which have to use a complicated Local NN to ensure feature sparsity, the adoption of XAI in AgileNN allows achieving higher feature sparsity with a much more lightweight Local NN and feature extractor. The inference latency of MCUNet is much higher (100-500ms) than that of other approaches, due to the complicated NN that is fully executed on the embedded device.

On the other hand, although edge-only inference incurs the minimum local computing delay, it suffers from the low wireless link rate at the local device[11] that results in a significantly higher wireless transmission latency due to the low data compressibility. The overall end-to-end latency of edge-only inference, hence, is higher than DeepCOD, SPINN and AgileNN.

Dataset	CIFAR-10	CIFAR-100	SVHN	ImageNet
Reduction	43.7%	15.8%	72.3%	20.8%

Table 2: Reduction of transmitted data size, compared to DeepCOD [65]

Such higher feature sparsity, on the other hand, also results in significant reduction on network transmission time. As shown in Table 2, such reduction on some datasets such as SVHN could exceed 70%. This reduction, even being lower than 20%, could be important in some IoT scenarios, where IoT devices are wirelessly connected to the 5G backbone network and will hence need to make usage-based payments to the 5G service provider [63].



(a) CIFAR-100 (b) SVHN

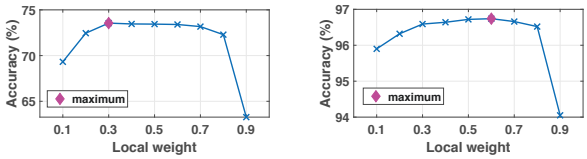
Figure 17: Accuracy with different compression rates

The Impact of Compression Rate. Since DeepCOD performs best among the three existing approaches for comparison, we further compare its performance with AgileNN when we apply different compression rates to transmit data features to the remote server. Results in Figure 17 over the CIFAR-100 and SVHN datasets show that, AgileNN can always achieve higher NN inference accuracy with the same compression rate being applied, due to its more agile and efficient enforcement of feature sparsity that results in better compressibility. In particular, when very high compression rates are applied, DeepCOD experiences significant accuracy reduction due to the limited representation power of its encoder, but such reduction in AgileNN is much lower.

The Impact of Prediction Reweighting. As described in Section 3.3, the predictions made by Local NN and Remote NN are combined towards the inference output, using a tuneable parameter α. Results

[11]Due to the local resource constraint, the maximum WiFi data rate at the STM32F746 MCU's WiFi module is capped at 6 Mbps.

in Figure 18 on the CIFAR-100 and SVHN datasets show that, the NN inference accuracy will significantly drop if highly biased values of α (e.g., close to 0 or 1) are being used. This is because using a very small α reduces the contribution of important features and could hence miss key information to inference. Increasing the value of α, on the other hand, imposes majority of the inference task to the Local NN, which may not be complicated enough to achieve high inference accuracy.



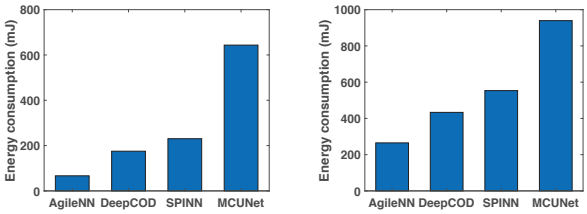
(a) CIFAR-100 (b) SVHN

Figure 18: Applying different weights

Instead, we conclude that the maximum inference accuracy can be achieved when α =0.3 for CIFAR-100 dataset and α =0.6 for SVHN dataset. Note that the optimal value of α is dependent on the data characteristics in the training dataset. In practice, this value can either be jointly trained offline with the feature extractor and NNs, or be manually tuned online based on the specific data characteristics for better inference accuracy.

7.3 Local Resource Consumption

In this section, we evaluate the amount of local resources at the embedded device that are consumed by AgileNN's inference. Such local resources include 1) the local battery power and 2) the local memory and flash storage. For fair comparison between different schemes, being similar with the previous experiments, we keep the gap between different schemes' inference accuracy to be within 5%.

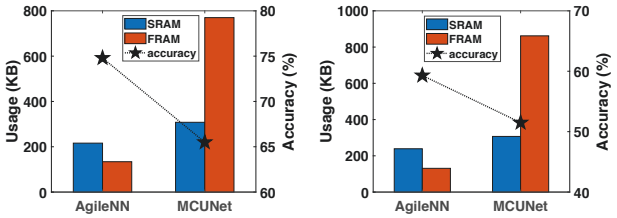


(a) CIFAR-100 (b) ImageNet-200

Figure 19: Local energy consumption per NN inference run

Energy consumption. We measure the amount of local device's energy consumption per NN inference run as the average over 100 inference runs. Such energy consumption includes both the local NN computing cost and data transmission cost via WiFi. As shown in Figure 19, since AgileNN uses a very lightweight feature extractor and local NN but achieves even higher feature sparsity with these lightweight NN structures, its runtime consumes less local energy in both computation and communication, leading to significantly higher energy efficiency. Especially when being used on smaller datasets such as CIFAR-100, its energy efficiency is at least 2.5x higher than that of DeepCOD, and is >8x higher than that of MCUNet.

Memory and storage usage. We measure the usage of on-board memory (SRAM) and storage (FRAM) by using the STM32Cube



(a) CIFAR-100 (b) ImageNet-200

Figure 20: Memory and storage usage

debugging software. As shown in Figure 20, due to the low complexity of feature extractor and NN, AgileNN's consumptions of the local device's memory and storage are both below 20%. In particular, when being compared with MCUNet whose NN structures are optimized via NAS, AgileNN occupies the similar amount of memory but a much smaller amount of external storage. Such high memory and storage efficiency is particularly important on weak embedded devices with very limited storage resources, because it allows deployment of much more powerful NN models on these devices and hence provide solid support to more challenging NN applications. On the other hand, the SRAM usages of DeepCOD and SPINN are at the similar level to that of AgileNN.

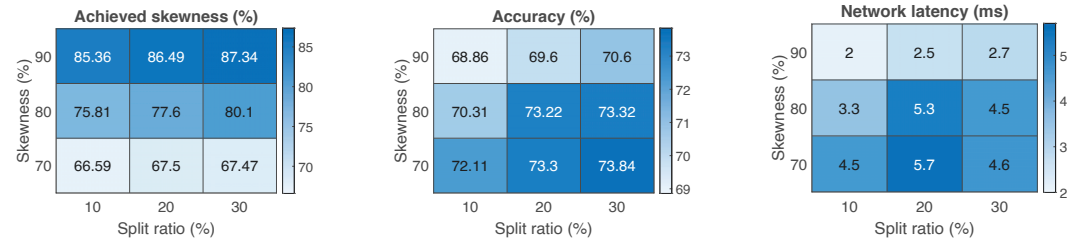
7.4 Effectiveness of Skewness Manipulation

Skewness manipulation is the cornerstone of efficient NN offloading in AgileNN. To investigate the effectiveness of AgileNN's skewness manipulation, we apply different requirements of feature importance skewness by varying the value of k between 3, 5 and 7, to retain 10%, 20% and 30% of features with the highest importance at the local NN. Correspondingly, we require the the normalized importances of these features to reach 70%, 80% and 90%, respectively.

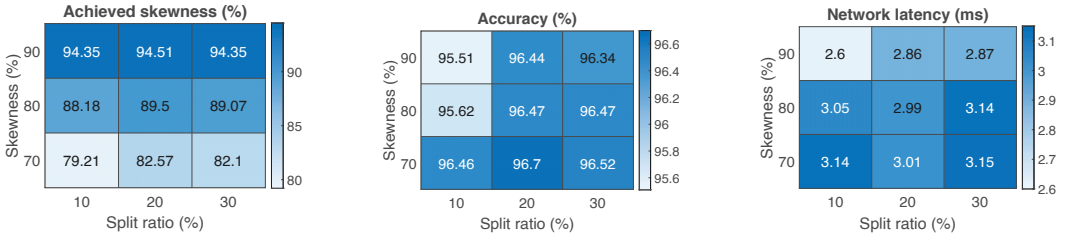
We first verify whether AgileNN's skewness manipulation can adequately achieve the required skewness in the extracted features. Figure 21(a) and 21(d) show that AgileNN can always meet the required skewness objective with minor difference. Especially on the SVHN dataset, the achieved skewness is even 4-12% higher than the objective. This demonstrates that our skewness loss function described in Section 3.1 is highly effective.

Second, Figure21(c) and 21(f) show that, with the same amount of important features being retained at the local NN, enforcing higher skewness on these features can increase the feature sparsity on the remaining less important features, hence reducing the network transmission latency. At the same time, such higher skewness also affects the NN inference accuracy as shown in Figure 21(b) and 21(e). The major reason is that, when the normalized importances of locally retained features are too high, the lightweight Local NN may not have sufficient representation power to correctly perceive these features, hence leading to extra accuracy loss. However, since the Local and Remote NNs are jointly trained, such accuracy drop can be always constrained within 3%.

These results demonstrate that AgileNN can effectively manipulate the skewness of feature importance in different settings, hence allowing flexible tradeoffs between the accuracy and cost of NN inference. Retaining more features at the local devices could help mitigate such accuracy drop, at the expense of extra local NN computations. In practice, the optimal choice of skewness requirement



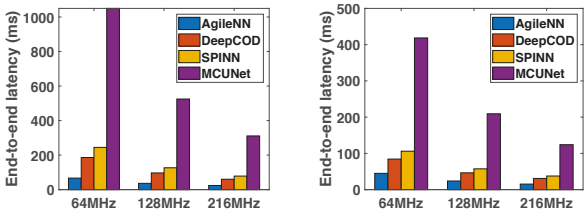
(a) CIFAR-100 achieved skewness (b) CIFAR-100 inference accuracy (c) CIFAR-100 network latency



(d) SVHN achieved skewness (e) SVHN inference accuracy (f) SVHN network latency

Figure 21: Effectiveness of skewness manipulation with different requirements of feature importance skewness

and split ratio will depend on the specific device's computation power and characteristics of the training dataset. We generally suggest that the optimal design choice is to retain 20% important features at the local device and require the normalized importance of these features to be >80%. Such skewness requirement will be used in all the following experiments.



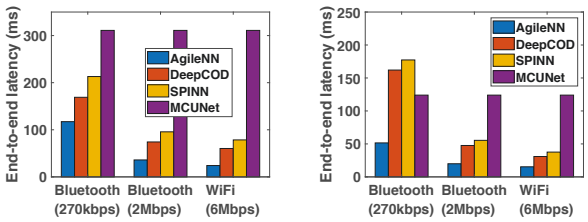
(a) CIFAR-100 (b) SVHN

Figure 22: The impact of different CPU Frequencies

7.5 Impact of Local CPU Frequency

Embedded devices may have CPUs with different frequencies. For example, the Arduino Nano uses an ATmega328 CPU at 16MHz and the STM32H743 MCU uses a dual-core ARM Cortex-M7 CPU at 480MHz, and the CPU frequency can also be adaptively configured at runtime. To study the impact of CPU frequency on AgileNN's performance, we adjust the CPU frequency of STM32F746 board by tuning its clock scaling factor. Here, we assume that most embedded devices, such as MCUs, will be exclusively used for NN inference when undertaking related computing tasks. Hence, we consider that the local device's CPU can be fully utilized for NN inference.

As shown in Figure 22, although the inference latency increases when the CPU frequency drops, such increase is always small when the CPU frequency drops from 216MHz to 64MHz, due to its lightweight feature exactor and Local NN. Comparatively, existing schemes suffer much higher performance degradation by running an expensive Local NN at the embedded device. For example, inference latency of MCUNet, SPINN and DeepCOD increased by 250%, 200% and 210%, respectively.



(a) CIFAR-100 (b) SVHN

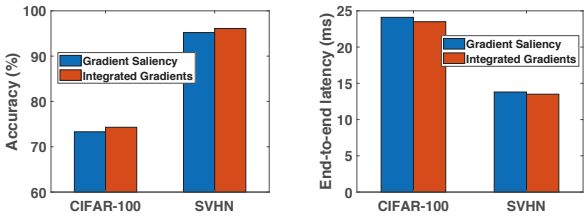
Figure 23: The impact of different wireless bandwidths

7.6 Impact of Network Bandwidth

Due to local constraints on power consumption and form factor, not all the embedded devices are equipped with high-speed WiFi modules. Instead, many of them have to use narrowband low-energy radios such as Bluetooth. Results in Figure 23 show that even when the available wireless network bandwidth is only 270kbps (95.5% lower than that of WiFi), AgileNN's high feature sparsity ensures that it can still restrain the NN inference latency to be 50ms on the SVHN dataset and 100ms on the CIFAR-100 dataset. In contrast, the inference latency of DeepCOD and SPINN is largely dependent on the wireless network bandwidth. These results imply that AgileNN outperforms other existing approaches in dynamic conditions of the wireless link connecting the local device and the server.

7.7 Choices of XAI techniques

The accuracy of importance evaluation varies with different XAI tools being used. To study such impact, we use two popular XAI tools: Gradient Saliency (GS) [16] and Integrated Gradients (IG) [59] to construct AgileNN. As shown in Figure 24, the performance of AgileNN remains stable with different XAI choices. IG makes



(a) Inference accuracy (b) End-to-end latency

Figure 24: Different XAI techniques

AgileNN perform slightly better because it aggregates more interpolations of NN outputs' gradients as described in Section 2.2. On the other hand, IG is more computationally expensive because it usually requires 20-100 times of gradient computations to obtain each importance measurement.

8 RELATED WORK

AI Attribution. AgileNN leverages current NN attribution tools to evaluate feature importance. Traditional attribution approaches apply random permutation [12] or zero masks [53] to specific input variables, and use the induced output variation to empirically indicate importance. Attention-based approaches [61, 68] embed a learning-based weighting layer into the NN, and the learned weights are used to indicate feature importance. However, these measurements are sensitive to different NN structures and cannot always ensure accurate evaluation.

Recent XAI techniques provide more accurate and robust attribution tools [56, 59]. They adopt NN output's gradients with respect to the input variables to derive importance, which is more fine-grained and can clearly tell in percentage how much each input variable contributes to the output value. XAI techniques are mainly used for analyzing data characteristics and understanding NN behavior, but its usage for improving offloading efficiency is rarely explored by the existing work.

On-device NN Inference. AgileNN is related to existing efforts on building lightweight NN models. NN compression [18, 25] and pruning [21, 27, 51] tailor complicated NNs by removing redundant weights and structures. Neural Architecture Search (NAS) [10, 44] pushes it to the theoretical limit by searching for the optimal NN structure under the NN complexity constraint. In certain circumstances where wireless connectivity is unavailable at the local embedded device and local inference is hence the only option, these techniques could be useful to support some simple NN inference tasks with low performance requirements. However, due to the extreme resource constraints on weak embedded devices, these techniques have limited capability in supporting more complicated NN inferences or achieving real-time NN inference.

AgileNN is also related to recent work of NN offloading. Early efforts transmit the compressed raw data to the server [45, 46]. To improve data compressibility, later work adopts a local NN that transforms the raw data into sparse features [20, 39, 42, 65], but the local NN should be complicated to ensure feature sparsity. Being orthogonal to AgileNN, there is work [30, 67] choosing to offload data to multiple servers to explore the heterogeneity of servers' computing power.

9 DISCUSSIONS

Reducing the training overhead. Using XAI to evaluate the feature importance is computationally expensive, due to frequent computation of gradients in every training iteration. A straightforward mitigation is to reduce the amount of such gradient computations, but this may affect the quality of skewness manipulation. Alternatively, since standard NN training also involves gradient operations, it's possible to reuse these existing gradients to speed up XAI evaluation. We also expect the AI community to develop more lightweight XAI techniques in the near future.

Extreme network conditions. As shown in Figure 23, AgileNN outperforms the existing schemes when the available network bandwidth is low. If the network is unavailable or encounters strong interference, AgileNN can still rely on the local predictor to make basic decisions. Because the most important features are undertaken by the local predictor, AgileNN makes the best effort to maintain inference accuracy. It is also viable to deploy more complicated local predictors to improve accuracy under such extreme conditions.

Other inference tasks. In evaluations of this paper, we mainly target image recognition tasks, but AgileNN can also be applied to other inference tasks such as video and audio analytics. In particular, due to the limit memory capacity at weak embedded devices, it may be difficult to take the entire video as one NN input (e.g., video summarization) if the video size is large, but instead the video could be split and analyzed in segments. Each video segment, then, can be processed in a per-frame basis on the local device, and the video analytic task hence falls back to an image recognition task. Similarly, audio data can be converted into a 2D spectrum, which can be treated as images for NN inference.

Offloading assisted training. Although AgileNN speeds up the AI inference on weak devices, it is hard for static NN models to adopt to new data and different application scenarios. Instead, the NN model should be promptly retrained at run-time with the new incoming data, while incurring the minimum computation costs. AgileNN can be possibly extended to online training by incorporating a federated learning framework [37], where multiple clients talk to a server without exposing local data. In this case, intermediate training results are forwarded to the server, which will then undertake majority of training overhead. Such extension of AgileNN will be our future work.

10 CONCLUSION

In this paper, we present AgileNN, a new technique that shifts the rationale of NN partitioning and offloading from fixed to agile and data-centric by leveraging the XAI techniques. AgileNN ensures real-time and accurate NN inference on extremely weak devices by migrating the required computations in NN offloading from online inference to offline training, and reduces the NN inference latency by up to 6× with similar accuracy compared to existing schemes.

ACKNOWLEDGMENTS

We thank the anonymous shepherd and reviewers for their comments and feedback. This work was supported in part by National Science Foundation (NSF) under grant number CNS-1812407, CNS-2029520, IIS-1956002, IIS-2205360, CCF-2217003 and CCF-2215042.

REFERENCES

[1] [n. d.]. Progressive Automations. https://www.progressiveautomations.com/pages/industrial-linear-actuators.

[2] [n. d.]. STM32F7/H7 Series Manual. https://www.st.com/resource/en/programming_manual/pm0253-stm32f7-series-and-stm32h7-series-cortexm7-processor-programming-manual-stmicroelectronics.pdf.

[3] Mohamed R Abdelhamid, Ruicong Chen, Joonhyuk Cho, Anantha P Chandrakasan, and Fadel Adib. 2020. Self-reconfigurable micro-implants for cross-tissue wireless and batteryless connectivity. In *MobiCom'20: Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*.

[4] Eirikur Agustsson, Fabian Mentzer, Michael Tschannen, Lukas Cavigelli, Radu Timofte, Luca Benini, and Luc V Gool. 2017. Soft-to-hard vector quantization for end-to-end learning compressible representations. *Advances in neural information processing systems* 30 (2017).

[5] Ejaz Ahmed, Michael Jones, and Tim K Marks. 2015. An improved deep learning architecture for person re-identification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3908–3916.

[6] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. 2016. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*. PMLR, 173–182.

[7] Roshan Ayyalasomayajula, Aditya Arun, Chenfeng Wu, Sanatan Sharma, Abhishek Rajkumar Sethi, Deepak Vasisht, and Dinesh Bharadia. 2020. Deep learning based wireless localization for indoor navigation. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. 1–14.

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[9] Anton Bakker and Johan H Huijsing. 1999. A low-cost high-accuracy CMOS smart temperature sensor. In *Proceedings of the 25th European Solid-State Circuits Conference*. IEEE, 302–305.

[10] Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul Whatmough. 2021. Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers. *Proceedings of Machine Learning and Systems* 3 (2021).

[11] Léon Bottou. 2012. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*. Springer, 421–436.

[12] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[13] Nam Bui, Nhat Pham, Jessica Jacqueline Barnitz, Zhanan Zou, Phuc Nguyen, Hoang Truong, Taeho Kim, Nicholas Farrow, Anh Nguyen, Jianliang Xiao, et al. 2019. ebp: A wearable system for frequent and comfortable blood pressure monitoring from user's ear. In *The 25th annual international conference on mobile computing and networking*. 1–17.

[14] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. 2018. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International Conference on Machine Learning*. PMLR, 794–803.

[15] Gioele Ciaparrone, Francisco Luque Sánchez, Siham Tabik, Luigi Troiano, Roberto Tagliaferri, and Francisco Herrera. 2020. Deep learning in video multi-object tracking: A survey. *Neurocomputing* 381 (2020), 61–88.

[16] Alex Davies, Petar Veličković, Lars Buesing, Sam Blackwell, Daniel Zheng, Nenad Tomašev, Richard Tanburn, Peter Battaglia, Charles Blundell, András Juhász, et al. 2021. Advancing mathematics by guiding human intuition with AI. *Nature* 600, 7887 (2021), 70–74.

[17] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[18] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*. 1269–1277.

[19] Estefania Munoz Diaz, Oliver Heirich, Mohammed Khider, and Patrick Robertson. 2013. Optimal sampling frequency and bias error modeling for foot-mounted IMUs. In *International Conference on Indoor Positioning and Indoor Navigation*. IEEE, 1–9.

[20] Amir Erfan Eshratifar, Mohammad Saeed Abrishami, and Massoud Pedram. 2019. JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services. *IEEE Transactions on Mobile Computing* 20, 2 (2019), 565–576.

[21] Biyi Fang, Xiao Zeng, and Mi Zhang. 2018. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. 115–127.

[22] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. 2020. Pruning neural networks at initialization: Why are we missing the mark? *arXiv preprint arXiv:2009.08576* (2020).

[23] Wei Gao, David Hsu, Wee Sun Lee, Shengmei Shen, and Karthikk Subramanian. 2017. Intention-net: Integrating planning and deep learning for goal-directed autonomous navigation. In *Conference on robot learning*. PMLR, 185–194.

[24] Graham Gobieski, Brandon Lucia, and Nathan Beckmann. 2019. Intelligence beyond the edge: Inference on intermittent embedded systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 199–213.

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

[26] Rick Groenendijk, Sezer Karaoglu, Theo Gevers, and Thomas Mensink. 2021. Multi-loss weighting with coefficient of variations. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 1469–1478.

[27] Song Han, Jeff Pool, John Tran, and William J Dally. 2015. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626* (2015).

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

[29] Robin Hesse, Simone Schaub-Meyer, and Stefan Roth. 2021. Fast axiomatic attribution for neural networks. *Advances in Neural Information Processing Systems* 34 (2021), 19513–19524.

[30] Ke-Jou Hsu, Ketan Bhardwaj, and Ada Gavrilovska. 2019. Couper: Dnn model slicing for visual analytics containers at the edge. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*. 179–194.

[31] Chuang Hu, Wei Bao, Dan Wang, and Fengming Liu. 2019. Dynamic adaptive DNN surgery for inference acceleration on the edge. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 1423–1431.

[32] Guosheng Hu, Yongxin Yang, Dong Yi, Josef Kittler, William Christmas, Stan Z Li, and Timothy Hospedales. 2015. When face recognition meets with deep learning: an evaluation of convolutional neural networks for face recognition. In *Proceedings of the IEEE international conference on computer vision workshops*. 142–150.

[33] Vikram Iyer, Vamsi Talla, Bryce Kellogg, Shyamnath Gollakota, and Joshua Smith. 2016. Inter-technology backscatter: Towards internet connectivity for implanted devices. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 356–369.

[34] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News* 45, 1 (2017), 615–629.

[35] Bryce Kellogg, Aaron Parks, Shyamnath Gollakota, Joshua R Smith, and David Wetherall. 2014. Wi-Fi backscatter: Internet connectivity for RF-powered devices. In *Proceedings of the 2014 ACM Conference on SIGCOMM*. 607–618.

[36] Jong Hwan Ko, Taesik Na, Mohammad Faisal Amir, and Saibal Mukhopadhyay. 2018. Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms. In *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE, 1–6.

[37] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492* (2016).

[38] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[39] Stefanos Laskaridis, Stylianos I Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas D Lane. 2020. SPINN: synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. 1–15.

[40] Ya Le and Xuan Yang. 2015. Tiny imagenet visual recognition challenge. *CS 231N* 7, 7 (2015), 3.

[41] Didier Le Gall. 1991. MPEG: A video compression standard for multimedia applications. *Commun. ACM* 34, 4 (1991), 46–58.

[42] Hongshan Li, Chenghao Hu, Jingyan Jiang, Zhi Wang, Yonggang Wen, and Wenwu Zhu. 2018. Jalad: Joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution. In *2018 IEEE 24th international conference on parallel and distributed systems (ICPADS)*. IEEE, 671–678.

[43] Yiran Li and Tong Zhang. 2011. Reducing dram image data access energy consumption in video processing. *IEEE Transactions on Multimedia* 14, 2 (2011), 303–313.

[44] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. 2020. Mcunet: Tiny deep learning on iot devices. *arXiv preprint arXiv:2007.10319* (2020).

[45] Zihao Liu, Tao Liu, Wujie Wen, Lei Jiang, Jie Xu, Yanzhi Wang, and Gang Quan. 2018. DeepN-JPEG: A deep neural network favorable JPEG-based image compression framework. In *Proceedings of the 55th annual design automation conference*. 1–6.

[46] Zihao Liu, Xiaowei Xu, Tao Liu, Qi Liu, Yanzhi Wang, Yiyu Shi, Wujie Wen, Meiping Huang, Haiyun Yuan, and Jian Zhuang. 2019. Machine vision guided 3d medical image compression for efficient transmission and accurate segmentation in the clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12687–12696.

[47] Yunfei Ma, Zhihong Luo, Christoph Steiger, Giovanni Traverso, and Fadel Adib. 2018. Enabling deep-tissue networking for miniature medical devices. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 417–431.

[48] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*.
[49] Mark R Nelson. 1989. LZW data compression. *Dr. Dobb's Journal* 14, 10 (1989), 29–36.
[50] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. 2011. Reading digits in natural images with unsupervised feature learning. (2011).
[51] Wei Niu, Xiaolong Ma, Sheng Lin, Shihao Wang, Xuehai Qian, Xue Lin, Yanzhi Wang, and Bin Ren. 2020. Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 907–922.
[52] Wei Niu, Xiaolong Ma, Yanzhi Wang, and Bin Ren. 2019. 26ms inference time for resnet-50: Towards real-time execution of all dnns on smartphone. *arXiv preprint arXiv:1905.00571* (2019).
[53] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. " Why should i trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 1135–1144.
[54] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. Model-agnostic interpretability of machine learning. *arXiv preprint arXiv:1606.05386* (2016).
[55] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4510–4520.
[56] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. 2017. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*. 618–626.
[57] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. 2017. Learning Important Features Through Propagating Activation Differences. *CoRR* abs/1704.02685 (2017).
[58] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[59] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *International Conference on Machine Learning*. PMLR, 3319–3328.
[60] Mingxing Tan and Quoc Le. 2021. Efficientnetv2: Smaller models and faster training. In *International Conference on Machine Learning*. PMLR, 10096–10106.
[61] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).
[62] Gregory K Wallace. 1992. The JPEG still picture compression standard. *IEEE transactions on consumer electronics* 38, 1 (1992), xviii–xxxiv.
[63] Dan Wang, Dong Chen, Bin Song, Nadra Guizani, Xiaoyan Yu, and Xiaojiang Du. 2018. From IoT to 5G I-IoT: The next generation IoT-based intelligent algorithms and 5G technologies. *IEEE Communications Magazine* 56, 10 (2018), 114–120.
[64] Zuxuan Wu, Xi Wang, Yu-Gang Jiang, Hao Ye, and Xiangyang Xue. 2015. Modeling spatial-temporal clues in a hybrid deep learning framework for video classification. In *Proceedings of the 23rd ACM international conference on Multimedia*. 461–470.
[65] Shuochao Yao, Jinyang Li, Dongxin Liu, Tianshi Wang, Shengzhong Liu, Huajie Shao, and Tarek Abdelzaher. 2020. Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. 476–488.
[66] Zhou Yu, Jun Yu, Yuhao Cui, Dacheng Tao, and Qi Tian. 2019. Deep modular co-attention networks for visual question answering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 6281–6290.
[67] Wuyang Zhang, Zhezhi He, Luyang Liu, Zhenhua Jia, Yunxin Liu, Marco Gruteser, Dipankar Raychaudhuri, and Yanyong Zhang. 2021. Elf: accelerate high-resolution mobile deep vision with content-aware parallel offloading. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*. 201–214.
[68] Hengshuang Zhao, Jiaya Jia, and Vladlen Koltun. 2020. Exploring self-attention for image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10076–10085.
[69] Wentao Zhao, Wei Jiang, and Xinguo Qiu. 2021. Deep learning for COVID-19 detection based on CT images. *Scientific Reports* 11, 1 (2021), 1–12.
[70] Yanbo Zhao and Zhaohui Ye. 2008. A low cost GSM/GPRS based wireless home security system. *IEEE Transactions on Consumer Electronics* 54, 2 (2008), 567–572.