Convolutional Neural Networks Inference Memory Optimization with Receptive Field-Based Input Tiling

Weihaio Zhuang (✉ zhuangweihaio@stu.kobe-u.ac.jp)

Kobe University: Kobe Daigaku <https://orcid.org/0000-0002-4765-0065>

Tristan Hascoet

Kobe University: Kobe Daigaku

Xunquan Chen

Kobe University: Kobe Daigaku

Ryoichi Takashima

Kobe University: Kobe Daigaku

Tetsuya Takiguchi

Kobe University: Kobe Daigaku

Yasuo Ariki

Kobe University: Kobe Daigaku

Research

Keywords: Neural Network, Memory Optimization, Receptive Field

Posted Date: August 10th, 2021

DOI: <https://doi.org/10.21203/rs.3.rs-743636/v1>

License: © ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

RESEARCH

Convolutional Neural Networks Inference Memory Optimization with Receptive Field-Based Input Tiling

Weihaio Zhuang*, Tristan Hascoet, Xunquan Chen, Ryoichi Takashima, Tetsuya Takiguchi and Yasuo Ariki

*Correspondence:

zhuangweihaio@stu.kobe-u.ac.jp
Kobe University, 1-1 Rokkodaicho,
Nada Ward, 657-0013 Kobe,
Japan

Full list of author information is
available at the end of the article

Abstract

Currently, deep learning plays an indispensable role in many fields, including computer vision, natural language processing, and speech recognition. Convolutional Neural Networks (CNNs) have demonstrated excellent performance in computer vision tasks thanks to their powerful feature-extraction capability. However, as the larger models have shown higher accuracy, recent developments have led to state-of-the-art CNN models with increasing resource consumption. This paper investigates a conceptual approach to reduce the memory consumption of CNN inference. Our method consists of processing the input image in a sequence of carefully designed tiles within the lower subnetwork of the CNN, so as to minimize its peak memory consumption, while keeping the end-to-end computation unchanged. This method introduces a trade-off between memory consumption and computations, which is particularly suitable for high-resolution inputs. Our experimental results show that MobileNetV2 memory consumption can be reduced by up to 5.3 times with our proposed method. For ResNet50, one of the most commonly used CNN models in computer vision tasks, memory can be optimized by up to 2.3 times.

Keywords: Convolutional Neural Network; Memory Optimization; Receptive Field

1 Introduction

Recent developments in computer vision have brought new models with improved accuracy. Unfortunately, higher accuracy often comes at the cost of increased resource consumption. Given their high resource consumption, state-of-the-art models are difficult to deploy in practical edge-device use-cases, in which resources are limited. Although it is possible to delegate the inference from edge devices to remote servers, this solution comes with additional network resource consumption, latency, and additional data privacy risks. Hence, optimizing the computing resources needed by CNN to run on edge devices without affecting the accuracy is a challenge of practical importance.

In this paper, we focus on one particular aspect related to the resource consumption of CNNs: we aim to optimize the memory consumption of CNNs during the inference phases. The memory used by a CNN model consists of two parts — activations and parameters. The number of parameters of the CNN is fixed when the model is designed. The amount of device memory used by activations is largely related to the resolution of the input. Figure 1 shows the ratio of the memory consumption caused by the activations to the memory consumption due to parameters

Figure 1: The ratio of activations memory consumption to the parameters at different resolutions in CNN inference

of the model for different resolutions; i.e., the memory consumption of the activations divided by the memory consumption of the parameters. As the input image resolution increases, the memory bottleneck in inference is gradually dominated by activations. For models with a smaller number of parameters, the effect of activations on memory consumption will become more pronounced as the resolution increases. For MobileNetV2 [1], the model with the smallest number of parameters among these models, the memory consumption for activations is 16 times higher than the parameters when the input resolution is 1024x1024.

Many computer vision benchmarks and implementations use image resolutions of 224x224 pixels, which is a common standard nowadays. This relatively low resolution explains why activations have not been considered a bottleneck in inference memory consumption, and most works aiming to compress networks for inference have focused on reducing the footprint of the model parameters. However, using higher resolution input is important for some practical computer vision tasks: Sabotke *et al.* [2] proposed that a CNN network trained with a large-size resolution has better performance on medical images. In industrial defect detection applications, the resolution of the image is often relatively high [3] [4]. Therefore, it would be beneficial to reduce the memory bottleneck caused by activations in the inference phase.

A straightforward method to reduce the memory consumption of CNN inference consists in sequentially processing input images in spatially arranged tiles, as illustrated in Figure 3a. That way, the computations of the model are equivalent to sequentially processing small resolution images, hence reducing the memory cost of inference. However, naively processing images in spatial tiles will result in approximate computations due to artifacts arising at the edges of each tile, which affects the model accuracy. These artifacts can be mitigated, and even completely removed using overlapping tiles to guarantee a computation exactly similar to the standard model execution. However, overlapping tiles introduce a computational cost overhead due to redundant computations at the edge of the tiles. This highlights a

fundamental trade-off between memory consumption and computations, which is the topic of this paper.

We propose to use the receptive fields of neurons at arbitrary layers to parameterize tiles that ensure that the output is unchanged and without unnecessary computation. Given these tiles, we are able to quantify the amount of redundant computation, and, hence, the memory overhead of a given choice of tiles. Similarly, we are able to compute the memory reduction achieved for different tiles. This provides us with formulas for calculating the trade-off between memory consumption and computational overhead later for different models and choices of tiling.

Several methods have been proposed to reduce memory consumption in inference. These can be broadly divided into three categories: model compression [5, 6, 7, 8], lightweight model design [9, 10, 1, 11, 12, 13, 14], and low-level operation implementation optimization [15, 16]. Most of these methods are carried out from the perspective of optimizing the memory consumption of the parameters.

Our method is based on using the receptive field to tile the input image, optimizing the memory bottleneck caused by activations during CNN inference. The method of calculating an arbitrary neural network receptive field proposed by Araujo *et al.* [17] provides us with tools for theoretical analysis. The method most related to our approach is proposed by Wu *et al.* [18], which also tiles the input to several parts and feeds them into the CNN model. However, this method does not use receptive field-based tiling, which prevents them from studying the exact trade-off we propose to study.

The remainder of this paper is organized as follows: We start by presenting prerequisite knowledge together with our methodology in Section 2.3. In Section 3 we use a toy parametrization of CNN architecture to derive the exact trade-off between memory consumption and computational overhead. Finally, in Section 4 we analyze the trade-off between computational overhead and memory consumption for some standard CNN architectures using our proposed method.

2 Proposed Method

Section 2.1 presents the background knowledge of the memory consumption of CNN models in the inference phase. The basics of the receptive field will be introduced in Section 2.2. Section 2.3 combines these concepts to formalize our proposed method.

2.1 Memory Consumption of CNN Inference

When a CNN performs inference, the total memory consumption is the sum of the parameters and activations stored in the device’s memory. Activations for inference are not accumulated. When the activations do not contribute to the subsequent inference, they are released from the memory. When executing a layer of the CNN model, the output activations from the previous layer are fed to the current layer, and then the input activations of the current layer are released from the memory because, at this point, the input activations are no longer contributing to the subsequent inference.

Table 1 and Figure 2 show the memory footprint of a simple CNN model at inference time. Table 1 shows the architecture of this CNN model. This model consists of six convolution layers and two max-pooling layers, with each two convolution layers

Table 1: Toy Network Architecture

Layer Name	Output Size (Channel, Height, Width)	Configuration (Kernel, Channel, Stride)
Input	3x32x32	-
Conv1, Conv2	32x32x32	3x3, 32, 1
Maxpool1	32x16x16	3x3, 2
Conv3, Conv4	64x16x16	3x3, 64, 1
Maxpool2	64x8x8	3x3, 2
Conv5, Conv6	128x8x8	3x3, 128, 1

Figure 2: Memory consumption footprint of toy CNN model inference

followed by one max-pooling layer, which is used to downsample the activations to one-half the length and width of the original. Figure 2 shows the memory consumption footprint of the model during the inference phase. The blue line represents the amount of device memory occupied by the model's parameters, and the parameters are always stored in the device memory. The orange curve shows the number of activations and the amount of memory occupied by the device by the parameters. At some point in the inference of the CNN model, the memory occupied by the activations and the parameters that have been stored in the device will be the peak of the memory consumption during the inference of the model, and our goal is to reduce this peak.

2.2 Receptive field of CNN models

This section introduces the basics of the receptive field related to this paper using a two-layer convolutional model and a CNN model with a residual block. The concepts and knowledge covered in this paper have been introduced by [17]. The receptive field (RF) is defined as the size of the region in the input that produces the feature. Figure 4a illustrates the effect of the number of layers on the RF size. The blue trapezoid on the left illustrates the case where the size of the RF is 5 after two layers of 3x3 convolution. That is, there are five elements in the one-dimensional input that contribute to the activations of the output. The part of the yellow trapezoid on the right illustrates the case where the RF is 3 after one layer of 3x3 convolution. Figure 4b illustrates the case where the RF has different values on two different branches

after using a residual block. A residual block can be seen as a module combining a branch with a 3x3 convolution and a branch with an addition operation, where the RF has a value of 3 after passing the 3x3 convolution, and for the addition operation, the RF is only one. The figure shows that the size of the RF increases as the number of layers of the CNN model network increases. In models with different branches,

Figure 4: The receptive field (RF) of the model with two layers of convolution and the model with residual blocks.

the calculated RF values are also different, and the maximum of the branches is used as the RF value. For example, in Figure 4b, we use the receptive fields of the 3x3 convolution as the receptive fields of the residual block. For different CNN models, the calculated RF values will also be different. The widely used CNN models usually consist of many connected convolutional layers, and the RF size calculated in the last layer of the model is usually larger than the resolution of the input image.

2.3 Proposed Method

The method proposed in this paper is shown in Figure 3b. We first determine the number of inputs after tiling the input image, which is 16 in Figure 3b, and then decide a position for the activations to be aggregated, so that the CNN model is divided into two parts: the head and the tail. The input image is tiled in accordance to the receptive field, and the black grids on the input image in Figure 3b show the size of each tile obtained after tiling. These tiles are fed one after another into the head of the CNN model, and the activations obtained from each tile are aggregated and fed into the tail of the model to complete the remaining inference. Note that in this paper, the aggregated activations are all the same size before being fed into the tail of the model. The reason we need to aggregate inside the network is that the peak memory consumption of CNN models usually occurs at the earlier layers, as shown in Figure 2. After the max-pooling operation, the memory occupied by the activations is significantly reduced as the length and width of the activations are downsampled to one half of the original size. If we can aggregate the activators after the max-pooling operation, i.e., after crossing the highest point of memory

consumption, we can reduce the memory consumption. However, max-pooling results in an exponential increase in RF, which results in excess computation, and we will discuss the trade-off between memory and computation in the next section on the position of activation aggregation.

3 Computation vs. Memory Trade-off

Our proposed method consists of feeding to the network a sequence of overlapping adjacent tiles. Hence the overlap between these tiles is fed into the model multiple times for computation, which introduces a computation redundancy overhead. For a particular CNN model, the computation/memory trade-off achieved depends on the specific layer at which the activations tiles are aggregated. In this section, we explore the memory and computational overhead trade-offs for sequential convolutional neural networks and convolutional neural networks with residual blocks, presenting formulas for theoretically calculating the computation and memory trade-offs of CNN models. Furthermore, we use a parametrizable CNN toy model to explore the influence of parameters on memory consumption and computation trade-offs, which provides a framework for designing more memory-friendly CNN models. We conclude this section with experiments on several successful CNN architectures to show the memory consumption and computation trade-offs they can achieve.

3.1 Sequential Convolutional Neural Network

First, we theoretically explore the peak memory consumption achieved by a sequential CNN model in the inference phase using RF-based tiling. Considering that our network has a total of n blocks, with i denoting the index of the block, $i = 1, 2, \dots, n$. In each block, there is a total of m modules composed of convolution, batch normalization, and ReLU, and j denotes the index of the module $j = 1, 2, \dots, m + 1$. H_j^i , W_j^i , and C_j^i represent the height, width, and channel, respectively, of the input activations of the j -th convolution module in the i -th block. Every two blocks are connected to each other using max-pooling. Figure 5 illustrates the architecture of this sequential CNN model. Then, the peak memory consumption of the inference is given by M , occupied by the input and output activations in a certain module plus the parameters R of the entire network. The peak memory consumption can then be calculated according to Equation 1. Our method optimizes the memory cost of activations, and the memory cost associated with the parameters R of the model do not change.

$$M = \max \left(H_i^j W_i^j C_i^j + H_i^{j+1} W_i^{j+1} C_i^{j+1} \right) + R \quad (1)$$

Moreover, we discuss the MAC (multiply-accumulate) required for the inference after using our proposed method in the sequence CNN model. The parameter k_j^i indicates the kernel size of the j -th convolutional layer in the i -th block. t denotes the number of tiled input images, and activations are aggregated after l block. The MAC Q can be calculated by Equation 2. The first summand of this formula expresses the amount of MAC required after tiles are fed into the head of the model, whereas the second part describes the MAC required for the model after the activations have been aggregated.

$$Q = t \sum_{i=1}^l \sum_{j=1}^m H_i^j W_i^j (C_i^j)^2 (k_i^j)^2 + \sum_{i=l+1}^n \sum_{j=1}^m H_i^j W_i^j (C_i^j)^2 (k_i^j)^2 \quad (2)$$

3.2 Residual Convolutional Block

Similar to 3.1, we assume that there are n residual blocks, and each block has m modules composed of convolution, batch normalization, and ReLU, as shown in Figure 6. The highest memory consumption M_r of the residual block after using our proposed method can be calculated by Equation 3, where R denotes the parameters of the entire model. H_i^j , W_i^j , and C_i^j are the input activations of each module, $i = 1, 2, \dots, n$, $j = 1, 2, \dots, m + 1$. H_i^1 , W_i^1 , and C_i^1 represent the input activations of the residual block, and H_i^{m+1} , W_i^{m+1} , C_i^{m+1} are the outputs of the last module of the residual block. Max-pooling is used to connect two residual blocks. The highest memory consumption of a residual block may occur in its last, that is, the moment before input activations and output activations are added, or when a module in the block is executed. Note that because the residual structure needs to add the input activations and output activations, the input activations will always stay in the device memory before completing the last step. In this paper, we only consider the aggregation of activations outside the residual block, although we can also do it inside the residual block, since the residual block requires the addition operation. It is necessary to keep the size of the added activations the same, which will increase extra memory consumption.

$$M_r = H_i^1 W_i^1 C_i^1 + \max \left(H_i^j W_i^j C_i^j + H_i^{j+1} W_i^{j+1} C_i^{j+1} \right) + R \quad (3)$$

Next, we discuss the increase in the amount of computation due to the tiling of the input image and the aggregating position. k_i^j represents the kernel size of each convolution module, and t symbolizes the number of the tiled input images. Suppose we perform the aggregation after l residual block. Q_r denotes the MAC of the CNN model after using our proposed method, calculated by Equation 4. Similar to Equation 2, the first half describes the computation of the head of the CNN model before the activations are aggregated, and the second half is the computation of the tail. In fact, Equations 4 and 2 are identical, and the residual structure does not change the way MAC is calculated after using this method.

$$Q_r = t \sum_{i=1}^l \sum_{j=1}^m H_i^j W_i^j (C_i^j)^2 (k_i^j)^2 + \sum_{i=l+1}^n \sum_{j=1}^m H_i^j W_i^j (C_i^j)^2 (k_i^j)^2 \quad (4)$$

3.3 Guidelines for Designing a More Memory-Friendly Model

In Sections 3.1 and 3.2, we presented theoretical formulations of the memory consumption and computation of the sequence CNN model and the CNN model with residual blocks after using our proposed method. In this section, we use a parametrizable toy model to analyze the trade-offs between memory and computational overhead. This analysis can provide us with tools to design memory-friendly CNN models.

The parameterizable toy model architecture is shown in Figure 7. The toy model is mainly composed of residual blocks, and it consists of m modules, which are connected by max-pooling layers, which downsamples the height/width of the activations to half of the original. The output channel C_o of each module is twice the input channel C_i of the module, and C_{st} is denoted as the number of channels of the first convolution. In each module, b residual blocks are connected, and each residual block is composed of n 3x3 convolutional layers. In the last residual block

(a) Memory consumption footprint of a toy model with settings $m = 3$, $b = 2$, $n = 3$, $C_{st} = 32$. (b) Memory consumption footprint of a toy model with settings $m = 3$, $b = 2$, $n = 4$, $C_{st} = 32$.

(c) The relationship between tiling size and computational redundancy overhead.

(d) The relationship between computational redundancy overhead and memory optimization.

Figure 8: Analysis of parametric toy models in terms of computational redundancy overhead and memory trade-off.

of the module, a 1×1 convolutional layer is used to increase the number of output channels.

Next, we study the impact of the different parameters of the toy model (number of convolutions in the residual block, aggregation layer index) on the memory / computational trade-off. Figure 8a and 8b illustrate the results for different parameters with the inputs being tiles of an original image of size 224×224 pixels and 16 tiles. Figure 8a shows the memory consumption of the model. The x-axis represents the order of execution of the layers: the left-most and right-most values represent the input and output layers respectively. The y-axis is the memory consumption (in bytes). Figure 8b is similar to Figure 8a, except for $n = 4$. The number of parameters of the $n = 4$ CNN model is higher due to the addition of one convolutional layer in each block, but with the same configuration of convolutional layers, the memory consumption for both configurations is the same after removing the parameters due to the activations.

The models in Figure 8a and Figure 8b each have six residual blocks, and the maximum memory consumption within these residual blocks in Figure 8a can be represented by P_i , $i = 1, 2, \dots, 6$, and the maximum memory consumption of each residual block in Figure 8b is given by $\hat{P}_j$, $j = 1, 2, \dots, 6$. We have eight positions for aggregating activations in these two models (six residual blocks and two max-pooling layers), and we denote these eight positions in Figure 8a by L_i , $i = 1, 2, \dots, 8$, where L_3 and L_6 are the positions for aggregating activations after the execution of the max-pooling. The eight positions in Figure 8b are represented by $\hat{L}_i$, $i = 1, 2, \dots, 8$. We can find that the peak memory usage is independent of n .

Both the curves in Figure 8c and Figure 8d can be calculated from the formulas presented in Sections 3.1 and 3.2. Figure 8c shows the change in the MAC relative overhead ratio for an increase of the tiling size as the activations are aggregated deeper (from L_1 or $\hat{L}_1$ to L_8 or $\hat{L}_8$). The MAC overhead ratio is the number of MAC after using our proposed method divided by the number of MAC before using our method. The blue and orange curves differ only in the aspect that the blue curve is for $n = 3$ and the orange curve is for $n = 4$. As we expect, as the aggregating position deepens, the tile size increases, the overlap between tiles increases, and the amount of redundant computation required also grows. When using $n = 4$, the receptive field calculated at the same residual block positions will be larger due to the fact that there are more layers of the model, and the input tiling required will be larger, with more demand for MAC. It is worth noting that the tiling size is the same at L_2 and L_3 , L_5 and L_6 . Even though the receptive field increases after one layer of max-pooling layers, the kernel size and stride of both max-pooling layers are 2. The same kernel size and stride values do not change the way the input image is tiled. We can consider a very simple example. Say we have 4 elements of one-dimensional input, after a ReLU layer. If we want to tile them into two tiles, then each part consists of two elements. If we add another layer of max-pooling with kernel size = 2 and stride = 2, then each tile still consists of two elements.

Figure 8d shows the memory relative optimization ratio as the MAC overhead ratio increases; the memory relative optimization ratio is the memory consumption before using our method divided by the memory consumption after using our method. Contrary to our expectations, the memory consumption does not become

more optimized as the computational redundancy increases. In the following, we summarize four cases where the memory optimization changes as the computational redundancy (aggregation position) varies. For that reason, we use Figure 8a as an example.

- 1) Memory relative optimization ratio equals 1 (aggregation position: L_1):
This happens when the aggregation position is before the peak of memory consumption in the default inference. P_2 is the peak of memory consumption in the default inference, and when the aggregation position is in L_1 , the activations after aggregation will still increase to P_2 without causing memory optimization.
- 2) Memory relative optimization ratio remains unchanged (aggregation position: from L_2 or L_3 to L_4):
This case is similar to the first case, and occurs when the memory consumption at the head of the model is smaller than the memory consumption at the tail of the model, and the memory consumption at the tail of the model does not change as the aggregation position deepens. When the activations are aggregated at L_2 , L_3 , and L_4 , although P_2 is crossed, P_4 becomes the new memory consumption peak. This change in position does not make the memory consumption more optimal, but L_4 is deeper than L_2/L_3 and requires more tiling, which increases the computational overhead.
- 3) Memory relative optimization ratio goes up (aggregation position: from L_1 to L_6):
This is what we expect to happen. As the tiling size increases, so does the MAC. This situation occurs across the current memory consumption peak. For example, when our aggregation positions go from L_1 to L_2 or L_3 , we cross P_2 , and increasing the optimization of memory consumption. When we cross P_4 from L_4 to L_5 or L_6 , the optimization of memory consumption will also increase.
- 4) Memory relative optimization ratio decreases (aggregation position: from L_5 or L_6 to L_8):
This happens when the memory consumed at the head of the model is larger than at the tail of the model. As the aggregation position deepens, the tiling increases, and the memory consumption of the input tiling in the head of the model becomes larger and larger in this case. For example, when going from L_5 or L_6 to L_7 , we may naturally think of P_6 as the memory bottleneck, but at this time, the memory consumption in the model head is already higher than P_6 as the tiling increases.

In Figure 8d, since the model with $n = 3$ has fewer layers than the model with $n = 4$, fewer tiles are required at the same aggregation position, thus requiring less MAC burden, and the curve for $n = 3$ is shifted to the left more than the curve for $n = 4$. The model with $n = 3$ also requires fewer parameters than the model with $n = 4$, and the bottleneck caused by the activations is more obvious. Therefore, the $n = 3$ curve is generally above the $n = 4$ curve.

Based on the above analysis, we give three guidelines for designing memory-friendly models, well suited for RF-based tiling memory optimization:

- 1) The maximum memory consumption of the model tends to occur at the relatively low layers. This way there is only a small amount of computational redundancy when aggregating the activations after crossing the memory consumption peak.
- 2) The memory used by the activations should be higher than the memory used by the parameters, so that the bottleneck will be more reflected in the memory occupied by the activations, and better memory optimization can be obtained with our method.
- 3) The number of convolutions in a residual block should be reduced appropriately because the method cannot aggregate activations in the residual block. The increase in the number of convolutions in the residual block will lead to more computational redundancy when aggregating activations behind the residual block and will result in more parameters.

4 Results and Discussion

We conducted experiments with our proposed method using some successful CNN models in an experimental environment with the Pytorch [19] deep-learning framework and Nvidia GPUs. Note that the optimization of device memory in the CNN inference phase depends to some extent on the optimization capabilities of the framework. Figure 9a illustrates the memory and MAC trade-offs using VGG16, ResNet18, ResNet50, and MobileNetV2 at 224x224 input resolution with a tiling number of 16. Figure 9b shows the experimental results for an input size of 1024x1024. The trade-off curves in Figure 9 end before the aggregation position reaches the classification layer of the model. MobileNetV2 is a lightweight network. The memory occupied by the parameters is much smaller than the memory occupied by the activations, so by optimizing the consumption of MobileNetV2 activations, it is possible to obtain a higher memory optimization rate than other models. In the case of 1024x1024 resolution input images, MobileNetv2 can achieve a memory optimization rate of 6.5 times, which means that the memory burden is only 19% that of the original one after using our method. By analyzing the trade-off curves in Figure 9, the trade-offs of the same CNN model at different resolutions are different. A higher resolution brings a larger memory bottleneck caused by activations, and the proposed method is more suitable for optimizing the memory bottleneck due to the high-resolution input. By comparing the two plots in Figure 9, the optimization rate of memory for 1024x1024 input is higher than that for 224x224 input under two resolutions of the same model. In both figures, we can find that the optimization rate of memory rises quickly with little MAC burden, which indicates that the memory bottleneck occurs at the lower layers when these models are performing default inference. The curves of both MobileNetV2 and ResNet50 in Figure 9a do not decrease, which indicates that the memory bottleneck occurs all the way to the tail of the model.

5 Conclusion

In this paper, we propose a method to reduce the memory consumption of convolutional neural networks in the inference phase that is orthogonal to other model compression methods. Using our method and the analytical results of this formulation, we can design a more memory-friendly model. We have used our proposed

(a) Computation and memory trade-offs for CNN models with 224x224 input.

(b) Computation and memory trade-offs for CNN models with 1024x1024 input.

Figure 9: Computation and memory trade-offs at different resolutions input after applying our proposed method to some widely used CNN models.

method in some successful CNN models, which reduces the memory consumption of the model in the device. However, it also imposes a certain computational overhead. Our method is more effective for input images with high resolutions.

Abbreviations

CNN: Convolutional Neural Network; RF: Receptive Field

Acknowledgements

The authors thank the editor and anonymous reviewers for their helpful comments and valuable suggestions.

Funding

Not applicable

Availability of data and materials

Please contact authors for data requests.

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

W.Z contributed to the presented research and writing of this paper. All authors read and approved the final manuscript

Author details

Kobe University, 1-1 Rokkodaicho, Nada Ward, 657-0013 Kobe, Japan.

References

1. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.-C.: MobileNetV2: Inverted residuals and linear bottlenecks. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520 (2018)
2. Sabottke, C.F., Spieler, B.M.: The effect of image resolution on deep learning in radiography. *Radiology: Artificial Intelligence* **2**(1), 190015 (2020)
3. Wang, T., Chen, Y., Qiao, M., Snoussi, H.: A fast and robust convolutional neural network-based defect detection model in product quality control. *The International Journal of Advanced Manufacturing Technology* **94**(9), 3465–3471 (2018)
4. Tao, X., Zhang, D., Ma, W., Liu, X., Xu, D.: Automatic metallic surface defect detection and recognition with convolutional neural networks. *Applied Sciences* **8**(9), 1575 (2018)
5. Krishnamoorthi, R.: Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342* (2018)
6. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015)
7. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. *Advances in neural information processing systems* **28**, 1135–1143 (2015)
8. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015)
9. Jin, J., Dundar, A., Culurciello, E.: Flattened convolutional neural networks for feedforward acceleration. *arXiv preprint arXiv:1412.5474* (2014)
10. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017)
11. Chollet, F.: Xception: Deep learning with depthwise separable convolutions. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1251–1258 (2017)
12. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5MB model size. *arXiv preprint arXiv:1602.07360* (2016)
13. Zhang, X., Zhou, X., Lin, M., Sun, J.: ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6848–6856 (2018)
14. Ma, N., Zhang, X., Zheng, H.-T., Sun, J.: ShuffleNet v2: Practical guidelines for efficient CNN architecture design. In: *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 116–131 (2018)
15. Gural, A., Murmann, B.: Memory-optimal direct convolutions for maximizing classification accuracy in embedded applications. In: *International Conference on Machine Learning*, pp. 2515–2524 (2019)
16. Wen, Y., Anderson, A., Radu, V., O'Boyle, M.F., Gregg, D.: Taso: Time and space optimization for memory-constrained DNN inference. In: *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pp. 199–208 (2020)
17. Araujo, A., Norris, W., Sim, J.: Computing receptive fields of convolutional neural networks. *Distill* **4**(11), 21 (2019)
18. Wu, S., Zhang, M., Chen, G., Chen, K.: A new approach to compute CNNs for extremely large images. In: *Proceedings of the 2017 ACM Conference on Information and Knowledge Management*, pp. 39–48 (2017)
19. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703* (2019)