# ConvReLU++: Reference-based Lossless Acceleration of Conv-ReLU Operations on Mobile CPU

Rui Kong[*]
kongrui@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Yuanchun Li[†]
Institute for AI Industry Research (AIR), Tsinghua University
Beijing, China

Yizhen Yuan
Institute for AI Industry Research (AIR), Tsinghua University
Beijing, China

Linghe Kong[†]
Shanghai Jiao Tong University
Shanghai, China

## ABSTRACT

Many activation values of Convolutional Neural Networks (CNNs) are zeros due to ReLU (Rectified Linear Unit), one of the most common activation functions used in modern neural networks. Since ReLU outputs are zero for all negative inputs, existing CNN acceleration approaches estimate zero outputs to skip redundant computation, which has to sacrifice accuracy for efficiency and leads to dilemma trade-offs and cockamamie configuration. In this paper, we introduce a lossless acceleration method ConvReLU++ for CNN inference on mobile devices, which accurately detects and skips zero-outputs for speedup without failures. The key to early negative detection is adopting reference-based upper-bounds calculation. This ensures that as soon as the intermediate results become negative, the final results are guaranteed to be negative. Upon detection, the remaining computation can be skipped and the following ReLU output can be simply set to zero. We rigorously prove the losslessness property of ConvReLU++, analyze the theoretical FLOPs reduction, and show the compatibility of our method with vector-level parallelism on mobile platforms. We implement ConvReLU++ in popular mobile inference frameworks and evaluate it on common deep vision tasks. The results demonstrate that ConvReLU++ can achieve 2.90% to 8.91% latency reduction over the original inference framework on edge devices without sacrificing accuracy. Our code can be found at https://github.com/GAIR-team/conv_relu_plus_plus.

## CCS CONCEPTS

• **Computer systems organization → Embedded systems**; • **Computing methodologies → Machine learning**.

## KEYWORDS

CNN inference, lossless acceleration, early negative detection, mobile CPU

**ACM Reference Format:**
Rui Kong, Yuanchun Li, Yizhen Yuan, and Linghe Kong. 2023. ConvReLU++: Reference-based Lossless Acceleration of Conv-ReLU Operations on Mobile CPU. In *The 21st Annual International Conference on Mobile Systems, Applications and Services (MobiSys '23), June 18–22, 2023, Helsinki, Finland.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3581791.3596831

## 1 INTRODUCTION

CNNs [26] are widely used in many computer vision tasks, such as optical character recognition [28, 33], object tracking [4, 22, 48, 54], industrial defect detection [49, 52] and medical image analysis [3, 41]. Deploying CNNs to edge devices (such as mobile phones, smart cameras, satellites, and robot arms) has become an increasing need due to various reasons including latency requirements, privacy concerns, and unavailable/unstable network connections [31, 50]. CPU is the most widely-used target hardware in these deployments due to its wide availablity and high compatibility [2, 5, 13, 15, 27].

The execution of typical CNNs requires a lot of computing power and energy. To alleviate the problem, various CNN inference acceleration approaches such as model compression [8, 44], domain-specific processors [42, 47], and system optimization [40, 55], etc., have been developed to execute CNNs more efficiently.

To optimize CNN inference on edge devices, we seek an opportunity for saving computation and latency from ReLU, a widely used activation function for CNNs due to its practicality. A nice property of the Conv-ReLU (Convolution layer followed by ReLU activation) structure is its high output sparsity, *i.e.,* the output of Conv-ReLU may contain a large portion of zeros. Specifically, the ReLU activation function will convert all negative values in the output of Convolution layer to zeros. It means that the computation cost to obtain the precise negative values in the Convolution operation may be wasted. Although such wasted computation may not induce too much overhead on high-performance machines where the dense matrix operations are highly optimized, it can be significant on the resource-limited edge and mobile devices.

Our method is based on the insight that *judging whether the output of a vector multiplication operation is negative can be faster than actually executing it. Thus, the vector multiplication operations before*

[*]This work was done while Rui Kong was an intern at the Institute for AI Industry Research (AIR), Tsinghua University.
[†]Corresponding authors: Yuanchun Li (liyuanchun@air.tsinghua.edu.cn), Linghe Kong (linghe.kong@sjtu.edu.cn). Yuanchun Li is also affiliated with Shanghai AI Laboratory.

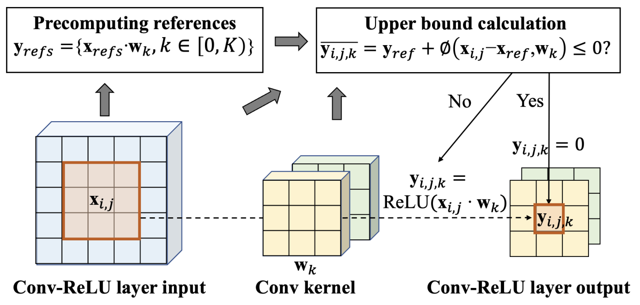
Precomputing references
$\mathbf{y}_{refs} = \{\mathbf{x}_{refs} \cdot \mathbf{w}_k, k \in [0, K)\}$
Upper bound calculation
$\overline{\mathbf{y}_{i,j,k}} = \mathbf{y}_{ref} + \emptyset(\mathbf{x}_{i,j} - \mathbf{x}_{ref}, \mathbf{w}_k) \leq 0?$
No
Yes
$\mathbf{y}_{i,j,k} = 0$
$\mathbf{y}_{i,j,k} =$ ReLU$(\mathbf{x}_{i,j} \cdot \mathbf{w}_k)$
$\mathbf{x}_{i,j}$
$\mathbf{w}_k$
$\mathbf{y}_{i,j,k}$
Conv-ReLU layer input
Conv kernel
Conv-ReLU layer output

**Figure 1: The high-level idea of ConvReLU++. In the Conv-ReLU structure, we can skip negative-output vector multiplications (*i.e.,* dot products between input patches $\mathbf{x}_{i,j}$ and convolution weights $\mathbf{w}_k$) with the help of carefully selected references.**

*a ReLU activation can be skipped if their output values are foreseen negative.* We call such cases of anticipating negative-value output without actual computation as *foreseen output sparsity.* Specifically, in the common Conv-ReLU structure, the interaction between a convolution filter and its receptive field is a vector multiplication operation where the optimization can be applied. Meanwhile, the limited parallelism and flexible instruction sets on mobile CPUs make it feasible to achieve actual speedups by skipping these negative-output computations.

Prior work has proposed various ways to identify and skip such redundant multiplications [6, 24, 46, 53, 56], but they are either lossy or only applicable in specific scenarios (*e.g.,* continuous inference with video content). The key to achieving acceleration on general vision tasks with foreseen output sparsity is identifying negative-output operations with low overhead and high success rate.

We introduce a reference-based method to identify and reduce unnecessary vector multiplications in Conv-ReLU structures for general vision tasks.

First, given a Conv kernel and its input patches to be multiplied, we perform a fast hash-based clustering to get a set of reference patches that are representative in the feature map. The dot products between the reference patches and the Conv kernel are pre-computed for comparison later. Then we calculate a tight upper-bound of the dot product between the Conv kernel and other input patches by comparing them with the reference patches. Finally, we identify the unnecessary (negative-output) multiplications and skip them in the convolution operation to reduce computational cost. The high-level idea is illustrated in Figure 1. All steps are designed to be compatible with vector-level parallelism, which can utilize advanced SIMD intrinsics on mobile platforms (*e.g.,* ARM NEON).

ConvReLU++ is implemented in *ncnn* [34] and TFLM (Tensorflow Lite for Microcontrollers) [11], two widely-used mobile deep learning frameworks. We also show that other similar structures (*e.g.,* Conv-BN-ReLU) and similar activation functions (*e.g.,* ReLU6) can be easily supported by extending our technique. Since our method is an operator-level optimization, we only need to modify the Conv-ReLU kernel in the frameworks. The models developed with the framework can be seamlessly executed without modification.

We evaluate our methods on image classification and object detection tasks with common datasets (ImageNet, TSRD, MNIST-ROT, Industrial Images, COCO, and PnPLO) and models (ResNet50, MobileNet, VGG16, Faster R-CNN, SSD, etc.). The results show that our methods can achieve up to 43.77% end-to-end FLOPs reduction on these tasks, constantly outperforming the original inference framework and strong baselines.

Our work makes the following research contributions:

- We introduce an operator-level lossless acceleration method for Conv-ReLU structures. Our approach is lossless, compatible with parallel computation, applicable to general vision inference tasks, and does not require model training.
- We design a novel hash-based clustering method for input reference selection and a lightweight upper-bound calculation method for redundant vector multiplication detection.
- We prove the losslessness of our method and demonstrate its effectiveness of FLOPs reduction and inference speedup on common tasks and real devices. Our code will be open-sourced.

## 2 BACKGROUND AND CHALLENGES

### 2.1 Preliminaries on Conv-ReLU Structures

Convolution neural networks (CNNs) are widely used in various vision inference tasks such as image classification, object detection, and semantic segmentation. Due to the remarkable performance achieved with a fairly small amount of parameters, it is also the default choice for intelligent mobile/edge vision applications.

Conv-ReLU structures (*i.e.,* convolutional layers followed by ReLU activation) are common in popular CNN models, which are used to convert the input image or feature map to higher-level features. Suppose $l_{\mathbf{W}}$ is a 2D convolution layer with input feature map $\mathbf{x}$, weight $\mathbf{W} \in \mathbb{R}^{R \times S \times C \times K}$, and bias $\mathbf{b} \in \mathbb{R}^K$, where $R \times S$ is the convolution kernel size and $C$ and $K$ are the sizes of input and output channels, respectively. We use $\mathbf{y}$ to represent the output of the Conv-ReLU layer. The element at location $(i, j)$ in the $k$-th channel of $\mathbf{y}$ can be written as

$$\begin{aligned}\mathbf{y}_{i,j,k} &= \text{ReLU}\left(\sum_{r=0}^{R-1}\sum_{s=0}^{S-1}\sum_{c=0}^{C-1} \mathbf{x}_{i+r,j+s,c} \cdot \mathbf{W}_{r,s,c,k} + \mathbf{b}_k\right) \\ &= \text{ReLU}\left(\mathbf{x}_{i,j} \cdot \mathbf{w}_k\right),\end{aligned} \tag{1}$$

where $\mathbf{x}_{i,j}$ and $\mathbf{w}_k$ are vectors of length $(CRS + 1)$ obtained by flattening the sub-matrix of $\mathbf{x}$ and $\mathbf{W}$ at corresponding indices, given by

$$\begin{aligned}\mathbf{x}_{i,j} &= \left[\mathbf{x}_{i,j,0}, \mathbf{x}_{i,j,1}, \ldots, \mathbf{x}_{i+R-1,j+S-1,C-1}, 1\right] \\ \mathbf{w}_k &= \left[\mathbf{W}_{0,0,0,k}, \mathbf{W}_{0,0,1,k}, \ldots, \mathbf{W}_{R-1,S-1,C-1,k}, \mathbf{b}_k\right],\end{aligned} \tag{2}$$

where we append the bias parameter $\mathbf{b}_k$ to the last of $\mathbf{w}_k$ and 1 to the last of $\mathbf{x}_{i,j}$ for simplicity.

Therefore, the whole computation in a Conv-ReLU structure can be seen as a batch of long-vector multiplications, each of which computes an element in $\mathbf{y}$, followed by a ReLU operation that converts all negative products to zeros. Since the vector multiplications are independent of each other, they can execute in parallel on multi-processor architectures.

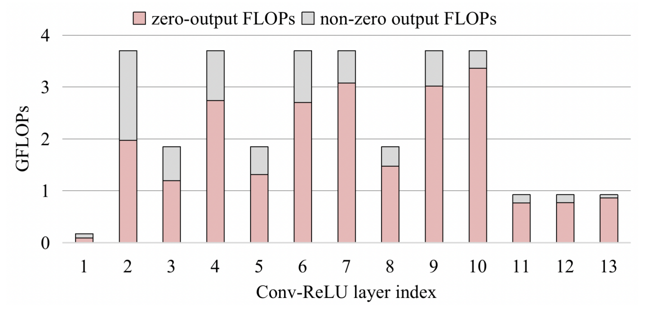

**Figure 2: The number of zero-output FLOPs in each Conv-ReLU layer of VGG16 by feeding 5000 random images in ImageNet.**

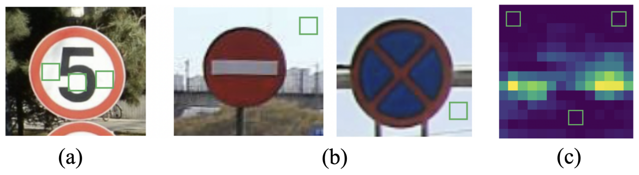

**Figure 3: Similarity between input patches of Conv-ReLU is common in the input image (a), across images in a batch (b), and in the feature map (c). Similar patches are marked in green boxes.**

## 2.2 Data Patterns of Conv-ReLU

We further analyze the input and output data patterns of Conv-ReLU structures by testing them on real vision inference tasks.

**High sparsity in the output.** According to Equation 1 and Equation 2, computing each element $\mathbf{y}_{i,j,k}$ of the output matrix of a Conv-ReLU layer requires a multiplication between two long vectors. However, the precise vector product is not used if the value is negative due to the ReLU operation. Since calculating whether $\mathbf{y}_{i,j,k}$ is positive or negative is relatively easier than actually calculating it, the computational cost of the long vector multiplication $\mathbf{x}_{i,j} \cdot \mathbf{w}_k$ can potentially be reduced if we can infer the sign of $\mathbf{y}_{i,j,k}$ with smaller cost.

The ratio of potentially reducible computation is related to the output sparsity of the Conv-ReLU operation (*i.e.,* the portion of zeros in the output matrix of each Conv-ReLU layer). Figure 2 shows the amount and portion of floating-point operations (FLOPs) the produce zeros in each Conv-ReLU layer of VGG model. We can notice that the sparsity ratio and the portion of computation related to the sparse output are both high (53.29% ~ 93.43%), meaning that there is a great potential for acceleration.

**High similarity between input patches.** As mentioned in Section 2.1, a Conv-ReLU operation can be viewed as a combination of many vector multiplications between a fixed Conv weight vector and many input patches at different locations. In a typical vision inference pass, the input patches of a Conv-ReLU structure may be similar or even identical with each other.

As shown in Figure 3, the similarity between input patches can be easily found in a single image, across different images, and in intermediate feature maps.

Patch similarities are common in most real-world applications, especially in high-resolution vision tasks, such as people tracking, medical diagnosis, material defect detection, etc. We have found that many previous approaches [35, 46, 53, 56] also utilize such patch similarity for deep learning acceleration.

## 2.3 Opportunity and Challenges

The high sparsity of output and high similarity between input patches in the Conv-ReLU structure bring us the opportunity to achieve lossless acceleration.

Specifically, if we can identify the negative-output vector multiplications in Conv-ReLU without calculating them, we have the opportunity to achieve lossless acceleration by skipping the unnecessary multiplications. Formally, suppose we have a function $\phi$ that calculates the upper-bound of a vector multiplication operation $\mathbf{x}_{i,j} \cdot \mathbf{w}_k$.

$$\overline{\mathbf{y}_{i,j,k}} = \phi(\mathbf{x}_{i,j}, \mathbf{w}_k) = upperbound(\mathbf{x}_{i,j} \cdot \mathbf{w}_k),$$

where the function $\phi$, the calculation of $\mathbf{y}_{i,j,k}$ can be written as

$$\mathbf{y}_{i,j,k} = \begin{cases} 0, & \text{if } \overline{\mathbf{y}_{i,j,k}} \leq 0 \\ \text{ReLU}\left(\mathbf{x}_{i,j} \cdot \mathbf{w}_k\right), & \text{otherwise.} \end{cases}$$

The computation of Conv-ReLU structures can be reduced if the upper-bound calculation function $\phi$ is more lightweight than the vector multiplication and the portion of $\mathbf{x}_{i,j}$ that has $\phi(\mathbf{x}_{i,j}, \mathbf{w}_k) \leq 0$ is high. Wakatsuki *et al.* [46] introduce such an upper-bound calculation function that can be used in continuous video inference scenarios. Specifically, they use the input patch difference between successive video frames to calculate the upper-bound:

$$\begin{aligned} \mathbf{x}_{i,j} \cdot \mathbf{w}_k &= \mathbf{x}_{i,j}^{t-1} \cdot \mathbf{w}_k + \left(\mathbf{x}_{i,j} - \mathbf{x}_{i,j}^{t-1}\right) \cdot \mathbf{w}_k \\ &\leq \mathbf{x}_{i,j}^{t-1} \cdot \mathbf{w}_k + ||\mathbf{x}_{i,j} - \mathbf{x}_{i,j}^{t-1}|| \times ||\mathbf{w}_k||, \end{aligned} \tag{3}$$

where $\mathbf{x}_{i,j}^{t-1}$ is the input patch at the same location of $\mathbf{x}_{i,j}$ in the last frame. Calculating the upper-bound with Equation 3 is lightweight since $\mathbf{x}_{i,j}^{t-1} \cdot \mathbf{w}_k$ has been computed in the last frame, $||\mathbf{w}_k||$ is constant, and $||\mathbf{x}_{i,j} - \mathbf{x}_{i,j}^{t-1}||$ can be reused for different kernels in an inference pass. However, their method can only be applied to video streams and requires the video content to be relatively static. The inference cost may even increase if the inter-frame difference is high.

In general vision inference tasks, using the last frame for comparison is not feasible, but the similarity between input patches of Conv-ReLU brings us another opportunity - we can let some input patches be the references for other similar input patches in the same forward pass. However, using this to achieve lossless acceleration is non-trivial, and there are three difficulties that have to be addressed:

(1) How to select the references in a single inference pass. The references must be useful and the selection must be efficient to achieve acceleration.
(2) How to effectively detect and skip unnecessary computations based on the selected references.
(3) How to retain the parallelism of Conv-ReLU operation, so that the acceleration method can be compatible with mobile SIMD architectures.

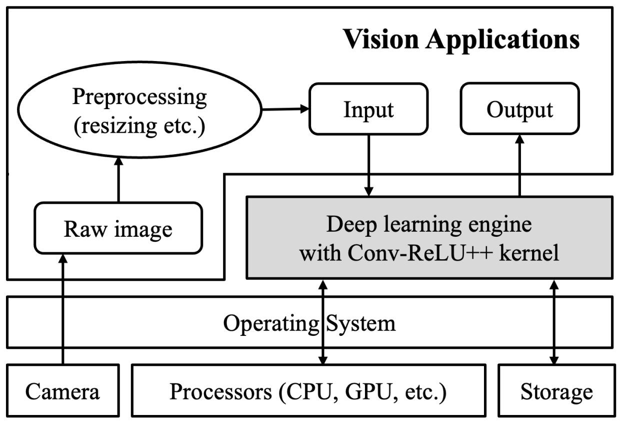
Vision Applications
Preprocessing (resizing etc.)
Input
Output
Raw image
Deep learning engine with Conv-ReLU++ kernel
Operating System
Camera
Processors (CPU, GPU, etc.)
Storage

**Figure 4: The vision inference system equipped with our method. Our method can be seamlessly integrated into the inference engine as an operator kernel.**

## 3 OUR APPROACH: CONVRELU++

We introduce ConvReLU++, a lossless acceleration method for mobile deep vision tasks. The main idea of our method is to skip unnecessary long-vector multiplications in Conv-ReLU operations based on the similarity between input patches. Specifically, we introduce a hash-based method to efficiently identify reference input patches, and a tight upper-bound calculation method to identify unnecessary vector multiplications. The whole solution is designed in compatibility with vector-level parallelism, which can map to actual latency reduction on mobile and edge platforms.

Our method can be transparently integrated into existing deep learning inference frameworks by replacing the Conv-ReLU kernel, as shown in Figure 4. As a result, our method has the key advantages that (1) no effort is required by the application developers and there will be no accuracy drop, and (2) the method is applicable to general CNN-based vision tasks as long as the model contains Conv-ReLU structures.

### 3.1 Overall Procedure

The overview of the Conv-ReLU operation optimized with ConvReLU++ is shown in Figure 5 and the pseudocode is shown in Algorithm 1. Similar to the original Conv-ReLU kernel, the input of ConvReLU++ kernel is the unfolded input image (or feature map), where each row is an input patch to be multiplied with the Conv kernel weight.

First, we introduce a patch hashing method to cluster similar input patches into groups. The clustering is efficiently achieved by directly computing the cluster indices *cluster_id* for all input patches with a lightweight hash function `patch_hash` (line 1).

Then we select the cluster centroids as the references (denoted as *references*) and the remained patches are denoted *non-references* (line 2-3). The Conv outputs of the references (*i.e.,* the dot products between the reference patches and the Conv kernel) are pre-computed before other input patches (line 5-7).

Next, for each non-reference input patch, $\mathbf{x}_{i,j}$ and Conv filter $\mathbf{w}_w$, ConvReLU++ calculates the upper-bound of their dot product

**Algorithm 1:** The procedure of the Conv-ReLU operation in ConvReLU++.

```
Input: Convolution layer input $\mathbf{x}$, layer weight $\mathbf{W}$, hash
       function $H$, pre-computed weight slice magnitudes
       w_MAG
Output: Conv-ReLU layer output $\mathbf{y}$
1  cluster_id = $round(\lambda \times patch\_hash(\mathbf{x}))$
2  references = [all cluster centroid indices]
3  non-references = [the remaining indices]
4  @parallelizable
5  for $(i, j)$ in references do
6      for $k = 0 \rightarrow K-1$ do
7          $\mathbf{y}_{i,j,k} = \mathbf{x}_{i,j} \cdot \mathbf{w}_k$
8  @parallelizable
9  for $(i, j)$ in non-references do
10     Get reference indice $(i', j')$ of cluster $\mathbf{cluster\_id}_{i,j}$
11     Get pre-computed $\mathbf{x}^{ref}_{i,j} = \mathbf{x}_{i',j'}$ and $\mathbf{y}^{ref}_{i,j} = \mathbf{y}_{i',j'}$
12     $\delta = ||\mathbf{x}_{i,j} - \mathbf{x}^{ref}_{i,j}||$
13     for $k = 0 \rightarrow K-1$ do
14         Calculate $\phi(\delta, \mathbf{w}_k)$ with $||\delta||$ and w_MAG$_k$
15         $\overline{\mathbf{y}_{i,j,k}} = y^{ref}_{i,j} + \phi(\delta, \mathbf{w}_k)$
16         if $\overline{\mathbf{y}_{i,j,k}} \le 0$ then
17             $\mathbf{y}_{i,j,k} = 0$
18         else
19             $\mathbf{y}_{i,j,k} = \mathbf{x}_{i,j} \cdot \mathbf{w}_k$
```

$\mathbf{y}_{i,j,k}$. The upper-bound calculation relies on a reference input $\mathbf{x}^{ref}_{i,j}$ and its pre-computed dot product with the same Conv filter $y^{ref}_{i,j}$ (line 10-15, discussed in Section 3.3). The reference $\mathbf{x}^{ref}_{i,j}$ we used here is the patch vector similar to $\mathbf{x}_{i,j}$, which lies in the same cluster with $\mathbf{x}_{i,j}$.

Finally, each time before actually computing $\mathbf{x}_{i,j} \cdot \mathbf{w}_k$ (line 19) , ConvReLU++ determines whether to skip the long-vector multiplication based on the calculated upper-bound $\overline{\mathbf{y}_{i,j,k}}$ (line 14). If $\overline{\mathbf{y}_{i,j,k}} \le 0$ (line 16), we can foresee that $\mathbf{x}_{i,j} \cdot \mathbf{w}_k \le 0$ and $ReLU(\mathbf{x}_{i,j} \cdot \mathbf{w}_k) = 0$, such that we can directly set $\mathbf{y}_{i,j,k} = 0$ without affecting the accuracy (line 17). Otherwise, we calculate the vector multiplication $\mathbf{x}_{i,j} \cdot \mathbf{w}_k$ to obtain the true value of $\mathbf{y}_{i,j,k}$ (line 19).

The final output of the Conv-ReLU operation $\mathbf{y}$ is the combination of the reference outputs, zeros produced by the skipped vector multiplications, and the results of non-skippable vector multiplications.

Next subsections will introduce the key steps in this procedure in more detail.

### 3.2 Hash-based Reference Selection

In the ConvReLU++ operator kernel, the first step is to select a set of reference input patches that will be compared with other input patches later for unnecessary operation detection.

The reference selection should meet several goals. First, the references should be representative of all input patches to ensure

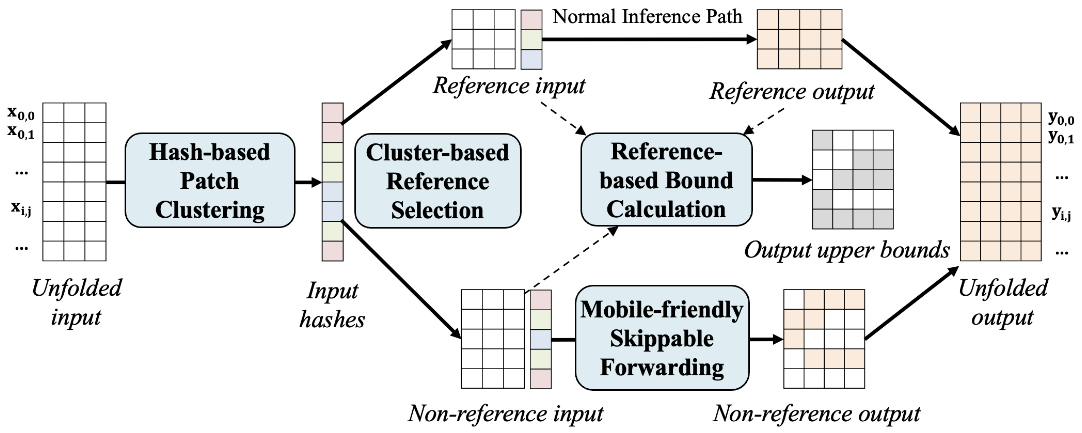

**Figure 5: The overall procedure of the Conv-ReLU operation in ConvReLU++.**

they are effective in later comparison. Second, the selection must be efficient, with the overhead much smaller than the corresponding Conv-ReLU computation. Third, the number of selected references should be controllable, since too many references would introduce a large overhead and too few references might be less effective.

Intuitively, a way to find representative inferences from a large set of input patches is clustering. Similar patches can be grouped together and cluster centroids can be selected as the references. However, applying traditional clustering method like k-means [17] to reference selection is infeasible since it requires to do an intensive comparison between input patches, which is too time-consuming at runtime.

Instead, we attempt to direct partition the input patches into groups through hashing. Specifically, we design a hash function `patch_hash` that maps each input patch to a hash id, and the input patches with the same hash id belong to the same cluster. Our method is similar to Locality Sensitive Hashing (LSH), but ours is more efficient as our hash function directly yields the cluster id with one step, instead of computing and transforming multiple bits in traditional LSH. We observe that the convolution kernels are able to extract input features, where similar input patches have similar output values. To make the hash function efficient, we convert the output values of the mean convolution kernel to integers as the hash indices. The selection of cluster centroid is not that critical, and we pick the first one as the centroid for simplicity. The detailed design is introduced below.

First, since the hash function is used in the Conv-ReLU kernel, we propose to directly use a lightweight Conv layer as the hash function. Suppose the shape of the original Conv kernel is of shape $R \times S \times C \times K$, where $R \times S$ is the kernel size and $C$ and $K$ are the input and output channel sizes. The Conv kernel shape of the hash function is set to $R \times S \times C \times 1$, so that the hash function can predict one value for each input patches in $\mathbf{x}$.

The effect of the hash function is equivalent to performing a linear transformation for each input patch $\mathbf{x}_{i,j}$, *i.e.*,

$$patch\_hash(\mathbf{x}_{i,j}) = w^{hash} \cdot \mathbf{x}_{i,j},$$

where $w^{hash}$ is a vector with the same length as an unfolded Conv filter $\mathbf{w}_k$. The weight of $w^{hash}$ is set to the mean of all Conv filters in the current Conv-ReLU structure so that the distance between the `patch_hash` values can reflect the similarity between the input patches and between their corresponding output values.

The output of `patch_hash` is a float number. We further convert it to a cluster id through scaling and rounding:

$$cluster\_id_{i,j} = round(\lambda \times patch\_hash(\mathbf{x}_{i,j})),$$

where $\lambda$ is a hyperparameter to control the cluster size. With this conversion, the similar input patches will have the same *cluster_id*.

Finally, we select the first input patch in each cluster as the cluster centroid, which is used as the reference patch for other patches in the same cluster. We save the mapping between the input patches and the cluster ids and the mapping between the cluster ids and the cluster centroids so that the reference for each input patch can be efficiently retrieved with $O(1)$ complexity. Meanwhile, the calculation of hash values only costs $1/C$ computation where $C$ is the output channel size of Conv-ReLU. As a result, our design achieves the goal of effectiveness and efficiency in reference selection.

## 3.3 Reference-based Bound Calculation

The goal of bound calculation in our system is to predict whether the dot product between an input patch and a Conv filter is negative so that the multiplication is skippable without accuracy sacrifice.

Our upper-bound calculation is based on

$$\mathbf{y}_{i,j,k} = \mathbf{x}_{i,j} \cdot \mathbf{w}_k \leq \mathbf{y}^{ref}_{i,j,k} + \phi(\mathbf{x}_{i,j} - \mathbf{x}^{ref}_{i,j}, \mathbf{w}_k), \tag{4}$$

where $\mathbf{x}_{i,j}$ is an input patch, and $\mathbf{x}^{ref}_{i,j}$ and $\mathbf{y}^{ref}_{i,j}$ are the input and output of its reference. Thus, we precompute the Conv outputs for all reference input patches before this step.

The key to using Equation 4 to calculate the upper-bound is to design the function $\phi$, which is used to calculate the upper-bound of the scalar product between $\mathbf{x}_{i,j} - \mathbf{x}^{ref}_{i,j}$ and $\mathbf{w}_k$. Let us abbreviate $\mathbf{x}_{i,j} - \mathbf{x}^{ref}_{i,j}$ to $\delta$ for simplicity.

The upper-bound function proposed in ConvReLU++ is:

$$\begin{aligned}\phi(\delta, \mathbf{w}_k) =&\delta[I_{\text{diff-sub}}] \cdot \mathbf{w}_k[I_{\text{diff-sub}}] \\ &+ ||\delta|| \times ||\mathbf{w}_k[I^c_{\text{diff-sub}}]||,\end{aligned} \tag{5}$$

where $I_{\text{diff-sub}}$ is a small subset of vector indices and $I^c_{\text{diff-sub}}$ is its complement. $\delta[I_{\text{diff-sub}}]$, $\mathbf{w}_k[I_{\text{diff-sub}}]$, and $||\mathbf{w}_k[I^c_{\text{diff-sub}}]||$ are subvectors of $\delta$ and $\mathbf{w}_k$ sliced by the corresponding indices. Next, we will show how and why Equation 5 gives a tight upper-bound of $\delta \cdot \mathbf{w}_k$.

First, by comparing the signs of the elements in $\delta$ and $\mathbf{w}_k$, we can obtain two subsets of vector indices $I_{\text{all}}$. $I_{\text{same}}$ is the indices where elements have the same sign, and $I_{\text{diff}}$ is the indices where the signs are different. *i.e.,*

$$I_{\text{same}} = \{i \mid \delta[i] \times \mathbf{w}_k[i] > 0\},$$

$$I_{\text{diff}} = \{i \mid \delta[i] \times \mathbf{w}_k[i] \le 0\}.$$

Then the dot product $\delta \cdot \mathbf{w}_k$ can be divided into two parts, including a positive part (the dot product of two same-sign subvectors) and a non-positive part (the dot product of two different-sign subvectors). Thus, we have

$$\begin{aligned}\delta \cdot \mathbf{w}_k =&\delta[I_{\text{same}}] \cdot \mathbf{w}_k[I_{\text{same}}] + \delta[I_{\text{diff}}] \cdot \mathbf{w}_k[I_{\text{diff}}] \\ \le&||\delta[I_{\text{same}}]|| \times ||\mathbf{w}_k[I_{\text{same}}]|| + \delta[I_{\text{diff}}] \cdot \mathbf{w}_k[I_{\text{diff}}].\end{aligned}$$

Comparing the signs of all elements of $\delta$ with $\mathbf{w}_k$ is time-consuming. In practice, we can only compare a small number (denoted as $E$, we set $E = 6$ by default) of indices where the elements of $\mathbf{w}_k$ have the largest magnitudes. Suppose $I_{\text{diff-sub}} \subseteq I_{\text{diff}}$ is a small subset of different-sign indices found by comparing the signs of $E$ largest-magnitude elements in $\mathbf{w}_k$ with the corresponding elements in $\delta$, and $I^c_{\text{diff-sub}} = I_{\text{all}} - I_{\text{diff-sub}}$ is the set of other indices. Then we have

$$\begin{aligned}\delta \cdot \mathbf{w}_k \le&||\delta[I^c_{\text{diff-sub}}]|| \times ||\mathbf{w}_k[I^c_{\text{diff-sub}}]|| \\ &+ \delta[I_{\text{diff-sub}}] \cdot \mathbf{w}_k[I_{\text{diff-sub}}].\end{aligned} \tag{6}$$

Since $I^c_{\text{diff-sub}}$ is a long list of indices, calculating the magnitudes of subvectors indexed by $I^c_{\text{diff-sub}}$ (*i.e.,* $||\delta[I^c_{\text{diff-sub}}]||$ and $||\mathbf{w}_k[I^c_{\text{diff-sub}}]||$) is still heavy. Fortunately, we can precompute $||\mathbf{w}_k[I^c_{\text{diff-sub}}]||$ with different combinations of $I_{\text{diff-sub}}$ (the number of combinations would not be large because $E$ is small) at offline. Moreover, we can replace $||\delta[I^c_{\text{diff-sub}}]||$ with its upper-bound $||\delta||$ in Equation 6, *i.e.,*

$$\begin{aligned}\delta \cdot \mathbf{w}_k &\le ||\delta|| \times ||\mathbf{w}_k[I^c_{\text{diff-sub}}]|| + \delta[I_{\text{diff-sub}}] \cdot \mathbf{w}_k[I_{\text{diff-sub}}] \\ &= \phi(\delta, \mathbf{w}_k).\end{aligned}$$

The upper-bound in Equation 3 adopted by Wakatsuki et al. [46] can be written as

$$\phi_{\text{base}}(\delta, \mathbf{w}_k) = ||\delta|| \times ||\mathbf{w}_k||. \tag{7}$$

Clearly, our upper-bound $\phi(\delta, \mathbf{w}_k)$ is tighter than $\phi_{\text{base}}(\delta, \mathbf{w}_k)$ because $||\delta|| \times ||\mathbf{w}_k[I^c_{\text{diff-sub}}]|| < ||\delta|| \times ||\mathbf{w}_k||$ and $\delta[I_{\text{diff-sub}}] \cdot \mathbf{w}_k[I_{\text{diff-sub}}]$ is negative.

To sum up, we can obtain $\phi(\delta, \mathbf{w}_k)$ with a lightweight sign comparison over a small number of indices ($I_{\text{diff-sub}}$), a dot product between two short vectors ($\delta[I_{\text{diff-sub}}] \cdot \mathbf{w}_k[I_{\text{diff-sub}}]$), a vector magnitude calculated shared by all convolution kernels ($||\delta||$), and several offline weight magnitude calculations ($||\mathbf{w}_k[I^c_{\text{diff-sub}}]||$). Finally, we can estimate the upper-bound of $\mathbf{x}_{i,j} \cdot \mathbf{w}_k$ according to Equation 4.

## 3.4 Theoretical Performance Analysis

We analyze the theoretical speedup of our method in this subsection. First, we consider the process of calculating the activation values of all convolution channels at a specific location $(i, j)$. The original convolution operation will require $K$ vector multiplications between the input vector $\mathbf{x}_{i,j}$ and the convolution kernel weights $\{\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_K\}$. Both $\mathbf{x}_{i,j}$ and $\mathbf{w}_k$ have the length $L = (CRS + 1)$. Thus, the total FLOPs of computing the convolution layer output at location $(i, j)$ with the conventional kernel is $KCRS$.

**Reduced Computation.** Suppose $p_\phi$ is the portion of convolution kernels whose dot product with the input $\mathbf{x}_{i,j}$ has a non-positive upper-bound (*i.e.,* $\overline{\mathbf{y}_{i,j,k}} \le 0$), then our Conv-ReLU kernel will skip $p_\phi K$ vector multiplications when computing the activation values $\{\mathbf{y}_{i,j,1}, \mathbf{y}_{i,j,2}, ..., \mathbf{y}_{i,j,K}\}$. The number of reduced FLOPs is $p_\phi KCRS$.

**Induced Overhead.** The overhead of ConvReLU++ comes from patch clustering and upper-bound calculation.

The computation FLOPs of $patch\_hash(\mathbf{x}_{i,j}) = w^{hash} \cdot \mathbf{x}_{i,j}$ is $NCRS$, where $N$ is the total number of $\mathbf{x}_{i,j}$. Line 2-3 takes no extra computation since it can be finished within line 1. Thus, the total overhead FLOPs of line 1-3 is $NCRS$.

Line 10-11 takes no extra computation since we have already computed them within line 1-3. The FLOPs of line 12 is $CRS$ which can be viewed as a vector product. Line 14-15 computes upper-bound $\overline{\mathbf{y}_{i,j,k}}$ where takes $E$ sign comparisons to determine the different-sign indices $I_{\text{diff-sub}}$ and about $E$ FLOPs to evaluate $\phi$. Thus, the total overhead FLOPs of line 10-15 is $N(CRS + KE)$.

ConvReLU++ can achieve speedup if the reduced computation is larger than the induced overhead, *i.e.,*

$$p_\phi KCRS - CRS - (CRS + KE) > 0.$$

In a regular inference pass, the ratio of skippable convolution kernels $p_\phi$ can easily satisfy $p_\phi K > 2$ with carefully selected references, and $KE$ is negligible since $E$ are small numbers. Thus, the speedup can be easily achieved.

The speedup ratio is determined by the quality of selected references and input data characteristics. The references used in our approach are the input patches with high similarity. Thus, ConvReLU++ can perform well on images with large same-content areas. In the ideal case, if we could find a perfect reference $\mathbf{x}^{ref}_{i,j}$ for each input vector $\mathbf{x}_{i,j}$ with $||\mathbf{x}_{i,j} - \mathbf{x}^{ref}_{i,j}|| \approx 0$, the ratio of skippable vector multiplications $p_\phi$ in our optimized kernel will reach the output sparsity ratio in each Conv-ReLU layer.

## 4 MOBILE PLATFORM INTEGRATION

The convolution operation is executed as a matrix multiplication in most inference frameworks, as shown in Figure 6. The input image (or feature map) is unfolded to a matrix with the *im2col* technique [9], where each row is an input patch $\mathbf{x}_{i,j}$ as described in Equation 2.

The Conv-ReLU operator kernel of ConvReLU++ requires skipping certain multiplications during the convolution operation, which breaks the integrity of matrix multiplication. Thus, it can hardly achieve meaningful acceleration on high-performance machines where the dense matrix multiplication is highly optimized (*e.g.,* with cuBLAS [36]).

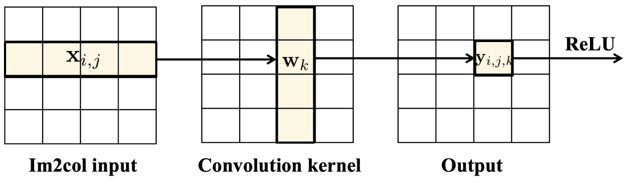

**Figure 6: A convolution operation is executed as a matrix multiplication between the unfolded input matrix x and unfolded convolution kernel matrix W.**

However, ConvReLU++ is especially suitable for mobile and edge devices with limited computational abilities and/or less parallel hardware architectures. Next, we describe how ConvReLU++ can be integrated into such devices.

**Devices with limited parallelism.** There are lots of embedded devices with little or no parallel computing ability [2, 5, 13]. They typically have only one processor without advanced instruction sets. On such devices, the convolution operator is a serial implementation, in which the floating point operations are executed sequentially. Our algorithm can be easily integrated into such devices according to Algorithm 1, where we only need to convert the extra vector operations to their naive implementations and insert them into the original operator.

**Devices with vector-level parallelism.** Some more advanced edge devices (*e.g.,* ARM-based Android smartphones) have the ability of vector-level parallelism, where the instruction set supports vector operations and multiple processors can perform the vector operations in parallel. Specifically, ARM has many vector registers for SIMD operations. For example, *vmul_f32(vec1, vec2, res)* is used for long-vector multiplications, and *vaddvq_f32(res)* is used to accumulate the result vector.

In ConvReLU++ kernel, there are lots of independent long-vector multiplications $\mathbf{x}_{i,j} \cdot \mathbf{w}_k$ within loops. Thus, we use the ARM intrinsics to implement the vector multiplications. Besides, we use multi-processing techniques (*e.g.,* OpenMP [10]) to concurrently perform the vector multiplications to achieve further acceleration. Because the operations in ConvReLU++ kernel (including reference precomputing, bound calculation, etc.) don't interface with each other, we can effectively utilize the parallel abilities on such devices.

**Devices with sparse-optimized hardware.** There are also some domain-specific processors that are proposed to support skippable matrix multiplication [1, 43, 51, 56, 57]. The main goal of these custom architectures is to accelerate SDDMM (Sampled Dense-Dense Matrix Multiplication [19]), in which the matrix multiplication is represented as $O = M_1 M_2 \odot S$, where $M_1$ and $M_2$ are input matrices, and $S$ is a boolean indicator matrix. $O[i][j]$ is not calculated where $S[i][j]$ is False.

ConvReLU++ is perfectly compatible with these custom architectures. Since the skippable Conv-ReLU operation in ConvReLU++ can be written as $ReLU(\mathbf{x}\mathbf{W} \odot S)$, where the values of $S$ at the indices with skippable multiplication are set to 0.

## 5 IMPLEMENTATION

We implement ConvReLU++ in two inference frameworks including *ncnn* [34], and TensorFlow Lite for Microcontrollers (TFLM for short) [11], which are the popular inference framework for mobile devices and tiny embedded devices respectively. In *ncnn*, we first calculate all `w_MAG` in advance before inference, which saves some time and space overhead. During inference, to efficiently implement the reference selection and upper-bound computations mentioned in Algorithm 1, we precompute all the $\delta$ and store references in a contiguous array to ensure CPU cache acceleration for subsequent upper-bound computations. For computing vector multiplications, we use SIMD acceleration and multi-threading to compute the upper-bounds and vector multiplications simultaneously. The implementation in TFLM is similar to in *ncnn*, except that the main logic is implemented as a serial version of Algorithm 1.

Next, we highlight several special cases we faced and handled during the implementation.

**Supporting BatchNorm (BN).** There are lots of Conv-BN-ReLU structures in modern CNNs, which are not directly supported using ConvReLU++. However, we can slightly adapt ConvReLU++ to support these structures. Specifically, we convert the Conv-BN-ReLU structure to the Conv-ReLU structure by merging the Convolution and BN into a single Convolution operation

$$\begin{aligned} \mathbf{y}_{i,j,k} &= \mathrm{ReLU}\left(\mathrm{BN}\left(\mathbf{x}_{i,j} \cdot \mathbf{w}_k + \mathbf{b}_k\right)\right) \\ &= \mathrm{ReLU}\left(\frac{\mathbf{x}_{i,j} \cdot \mathbf{w}_k + \mathbf{b}_k - \mu}{\sqrt{\sigma^2 + \epsilon}}\gamma + \beta\right) \\ &= \mathrm{ReLU}\left(\mathbf{x}_{i,j} \cdot \mathbf{w}'_k + \mathbf{b}'_k\right), \end{aligned}$$

where $\mathbf{w}'_k = \frac{\gamma}{\sqrt{\sigma^2+\epsilon}}\mathbf{w}_k$ and $\mathbf{b}'_k = \frac{\mathbf{b}_k - \mu}{\sqrt{\sigma^2+\epsilon}}\gamma + \beta$. Thus, we can replace all Conv-BN-ReLU structures with Conv-ReLU that can be directly accelerated by ConvReLU++.

**Supporting ReLU6.** ReLU6 is a modification of the ReLU where we limit the activation to a maximum size of 6. This is due to increased robustness when used with low-precision computation. Many CNN models designed for mobile platforms (*e.g.,* MobileNet) utilize this structure. We support ReLU6 by adjusting the way to compute $\mathbf{y}_{i,j,k}$, where we can use both the upper-bounds and the lower bounds to determine skippable vector multiplications.

$$\mathbf{y}_{i,j,k} = \begin{cases} 0, & \text{if } \overline{\mathbf{y}_{i,j,k}} \le 0 \\ 6, & \text{if } \underline{\mathbf{y}_{i,j,k}} \ge 6 \\ \mathrm{ReLU}\left(\mathbf{x}_{i,j} \cdot \mathbf{w}_k\right), & \text{otherwise,} \end{cases}$$

where $\overline{\mathbf{y}_{i,j,k}}$ is the upper-bound and $\underline{\mathbf{y}_{i,j,k}}$ is the lower bound. Following the procedure of upper-bound calculation in ConvReLU++, we can similarly calculate the lower bound by

$$\begin{aligned} \underline{\mathbf{y}_{i,j,k}} = \mathbf{x}_{i,j} \cdot \mathbf{w}_k &= \mathbf{x}^{ref}_{i,j} \cdot \mathbf{w}_k - \left(\mathbf{x}^{ref}_{i,j} - \mathbf{x}_{i,j}\right) \cdot \mathbf{w}_k \\ &\le \mathbf{x}^{ref}_{i,j} \cdot \mathbf{w}_k - \phi(\mathbf{x}^{ref}_{i,j} - \mathbf{x}_{i,j}, \mathbf{w}_k). \end{aligned}$$

## 6 EVALUATION

We evaluate ConvReLU++ to answer the following research questions:

(1) Is our approach able to reduce FLOPs of CNN inference?
(2) How effective is each component in our system? *e.g.,* hash-based reference selection and reference-based bound calculation.

(3) Does the mobile platform integration bring actual latency reduction?
(4) What's the memory overhead of our method?

## 6.1 Experimental Setup

We select common and real-world deep vision tasks to evaluate our method, in comparison with closely-related baselines.

**Tasks, datasets, and models.** We focus on image classification tasks and object detection tasks, which are the most common tasks in literature and practice. The datasets we used include standard datasets, such as MNIST-ROT (an extended version of MNIST) [25], ImageNet [12], TSRD (Traffic Sign Recognition Database) [58], and industrial defect detection dataset (Industrial Images for short) [45]. Regarding the models, we include common and state-of-the-art CNN models including ResNet, VGG, FasterRCNN, MobileNet, etc. for complex tasks and a vanilla two-layer CNN model (VanillaCNN for short) for simple tasks. We use the standard architectures and pre-trained weights for the models without any customization. The detailed list is shown in Table 1.

**Baselines.** It is difficult for us to find a baseline that is both lossless and generic. Instead, we compare with SparseNN [56] and SeerNet [6], which are representative lossy approaches that also propose to skip unnecessary or unimportant computation in Conv-ReLU structures. Wakatsuki et al. [46] is lossless, but it supports video scenario only. Thus we just compare our upper-bound calculation method $\phi$ with theirs ($\phi_{base}$). Unfortunately, we couldn't find the source code of these baseline methods, thus we implement them by ourselves following their papers. Since supporting different models with the baseline methods requires tremendous effort, we compare with them the vanilla two-layer CNN model and the MNIST-ROT dataset.

**Platforms.** We also test the latency reduction performance of ConvReLU++ on real edge devices, including an Arduino Nano board (Arm Cortex-M4) and an Android smartphone (Snapdragon 865), which are based on TFLM [11] and *ncnn* [34] frameworks respectively.

## 6.2 Inference FLOPs Reduction

We first evaluate the effectiveness of inference FLOPs reduction, which is a direct indicator of how much computational cost can be saved. The results are shown in Table 1.

First of all, ConvReLU++ achieves lossless FLOPs reduction on all tasks, with the reduction ratio ranging from 1.87% to 43.77% according to the results. Given the fact that our method is a lossless approach, we believe our method is useful for a wide range of deep vision tasks.

The computation reduction ratio of our method is highly dependent on both the dataset and the model. We first analyze the impact of datasets. The highest FLOPs reduction (43.77%) is achieved on the image classification task with the MNIST-ROT dataset. The Industrial Images and TSRD datasets also produce high FLOPs reduction ratios of 24.91% and 12.16%. However, the FLOPs reduction is less significant (lower than 6%) on ImageNet. A similarity of the datasets on which ConvReLU++ performs well is that the images contain large areas of similar pixels (*e.g.,* background, object surface, etc.), which helps our method to select betters references and skip more vector multiplications. Many other real-world data have similar properties as MNIST images, such as medical images and space images. We believe ConvReLU++ can also perform well on such tasks.

Then we analyze the impact of model architectures on the effectiveness of ConvReLU++. On the ImageNet dataset, ResNet50, VGG16, and SqueezeNet achieve different ratios of FLOPs reduction ranging from 2.62% to 5.28%. On the Industrial Images dataset, ResNet50 achieves 11.81% FLOPs reduction and VGG16 achieves 24.91% FLOPs reduction. Our method constantly performs better with VGG16 than ResNet50. One reason is that VGG16 contains a larger portion of Conv-ReLU structures where our method can be applied. Besides, ResNet50 model is deeper, and the spatial locality in deeper layers is low as the features are mixed together, which degrades the ability of our method in detecting skippable vector multiplications.

The types of vision tasks do not pose unique challenges for our approach. The FLOPs reduction patterns we observed on object detection tasks are similar to those on the classification tasks. However, since the ratio of Conv-ReLU structures is lower in detection models, the speedup ratio is not as high as on classification tasks.

We further conduct a breakdown analysis of the FLOPs reduction in each Conv-ReLU layer. The results are shown in Figure 7, where we can observe that almost all Conv-ReLU layers can benefit from our methods for FLOPs reduction. However, the ratio of FLOPs reduction is generally lower in deeper layers, which explains the reason why our method performs better on shallower networks.

**Comparison with the lossy method.** Table 2 shows the computation reduction performance of our methods and the baselines (SparseNN & SeerNet). The baselines have to trade accuracy for efficiency. Although they can sometimes achieve higher speedup, it usually leads to higher accuracy loss, even on simple datasets like MNIST-ROT. We suspect the accuracy loss may become even higher on more complicated tasks. Moreover, finding the optimal tradeoff between accuracy and latency is also difficult and time-consuming for developers.

## 6.3 Breakdown Analysis

We further analyze the effectiveness of each individual component in ConvReLU++.

**Reference Selection.** We compare our hash-based reference selection method with two baselines, including a stride selection strategy that uses the input patch $\mathbf{x}_{i,j}$ as the reference for patches between $\{\mathbf{x}_{i,j-k}$ and $\mathbf{x}_{i,j+k}, ...\}$ ($k$ is the stride size) and a random strategy that selects random input patches as the references. Table 3 shows the relative FLOPs achieved by ConvReLU++ using the three different reference selection strategies.

The best reference effectiveness (56.23% relative FLOPs, aka. 43.77% FLOPs reduction) is achieved by our hash-based strategy with a reference ratio of 6.72%. This is due to the ability of our method to cluster similar patches through efficient hashing. The random reference selection strategy is not effective as it can only produce 10% FLOPs reduction with different reference ratios. The stride strategy is better because it utilizes some simple spatial locality patterns, although the effectiveness is still much lower than ours.

**Table 1: Models and tasks used in our experiments and the average FLOPs reduction ratio achieved by our approach. The # Layers column is the number of Conv-ReLU layers compared with the total number of layers (excluding non-parametric layers).**

| ID | Task | Model | # Layers | Dataset | Original GFLOPs | Our GFLOPs |
|---|---|---|---|---|---|---|
| 1 | Classification | VanillaCNN | 2/2 | MNIST-ROT [25] | 0.02 | 0.01 (-43.77%) |
| 2 | Classification | ResNet50 [18] | 33/50 | ImageNet [12] | 8.24 | 8.02 (-2.62%) |
| 3 | Classification | VGG16 [32] | 13/16 | ImageNet [12] | 15.50 | 14.87 (-4.08%) |
| 4 | Classification | SqueezeNet [21] | 12/18 | ImageNet [12] | 0.35 | 0.33 (-5.28%) |
| 5 | Classification | ResNet50 [18] | 33/50 | Industrial Images [45] | 24.22 | 21.40 (-11.81%) |
| 6 | Classification | VGG16 [32] | 13/16 | Industrial Images [45] | 90.64 | 68.06 (-24.91%) |
| 7 | Detection | FasterRCNN [39] | 8/17 | COCO [29] | 23.50 | 21.81 (-7.21%) |
| 8 | Detection | FasterRCNN [39] | 8/17 | TSRD [58] | 23.50 | 20.64 (-12.16%) |
| 9 | Detection | MobileNet-SSD [20] | 47/60 | COCO [29] | 1.23 | 1.16 (-5.44%) |
| 10 | Detection | MobileNet-SSD [20] | 47/60 | TSRD [58] | 1.23 | 1.10 (-10.82%) |

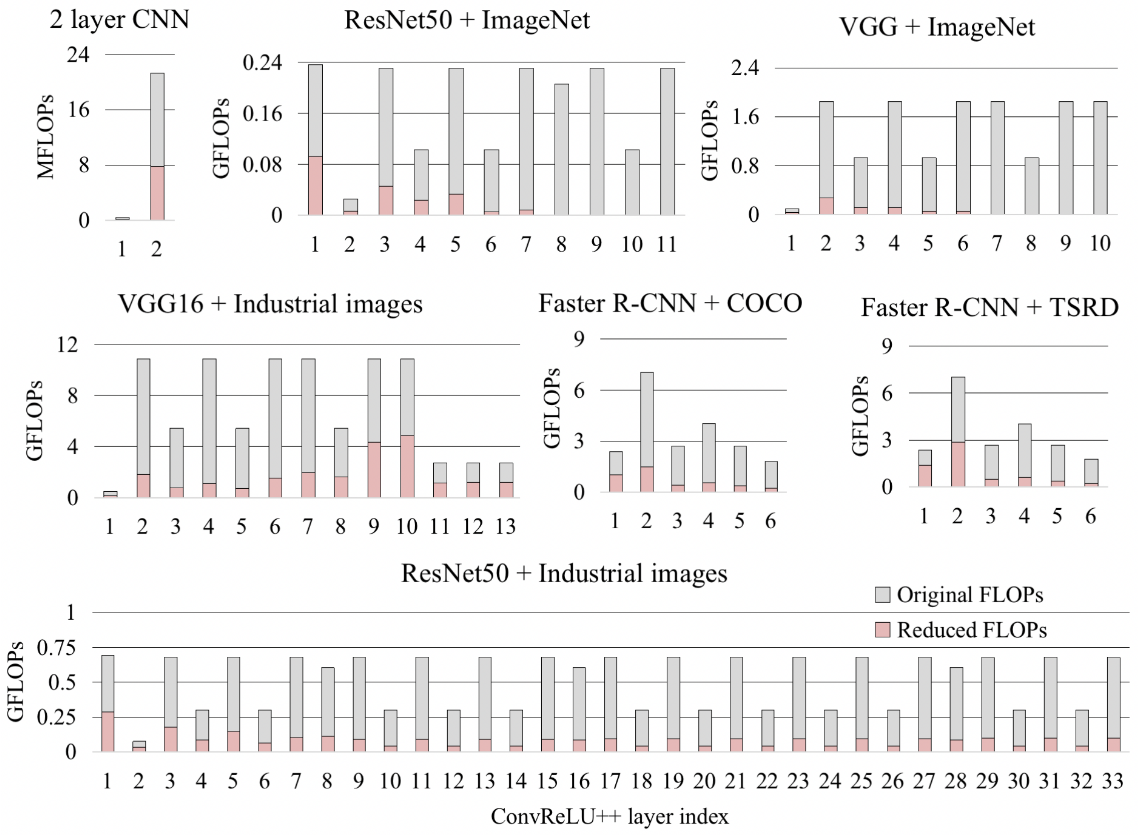

**Figure 7: Layer-wise FLOPs reduction achieved by ConvReLU++ on different models and tasks.**

The reference ratio (the portion of input patches used as references) of our method is controlled by the hash scaling factor $\lambda$ (Section 3.2). We can see that the FLOPs reduction effectiveness is not significantly affected by the reference ratio, but there exists an optimal reference ratio. Other reference ratios higher or lower than it may lead to higher reference precomputing overhead or lower success rate of negative output detection.

We also visualize the cluster indices generated by ConvReLU++ at different layers in Figure 8. We can see that similar patches are likely to have the same cluster indices, which demonstrates the effectiveness of our patch clustering. Meanwhile, we observe that the similarity between cluster indices is lower for high-variance images and in deeper layers, which can explain the difference of FLOPs reduction performance across models and datasets in Section 6.2.

**Table 2: Relative FLOPs of our method & lossy baselines on MNIST-ROT dataset with Vanilla CNN model. r is a hyper-parameter of SparseNN. * means that SeerNet needs to run a whole 4-bit quantized model.**

| Method | Test Accuracy | Relative FLOPs |
|---|---|---|
| SparseNN with r=1 | 14.82% | 17.23% |
| SparseNN with r=2 | 42.35% | 35.75% |
| SparseNN with r=4 | 72.88% | 68.31% |
| SparseNN with r=9 | 93.34% | 144.09% |
| SeerNet | 90.12% | 27.44% + 100.00% (4-bit)* |
| Ours | 93.34% | 56.23% |
| Vanilla CNN | 93.34% | 100.00% |

**Table 3: Relative FLOPs achieved by ConvReLU++ with different reference selection strategies. Results obtained with VanillaCNN and MNIST-ROT.**

| Method | Relative FLOPs | Reference Ratio |
|---|---|---|
| hash selection | 56.23% | 6.72% |
| hash selection | 58.24% | 1.09% |
| hash selection | 61.09% | 12.37% |
| stride selection | 78.42% | 50.00% |
| stride selection | 79.19% | 33.33% |
| stride selection | 83.44% | 25.00% |
| random selection | 88.19% | 5.00% |
| random selection | 90.12% | 15.00% |
| random selection | 89.35% | 25.00% |

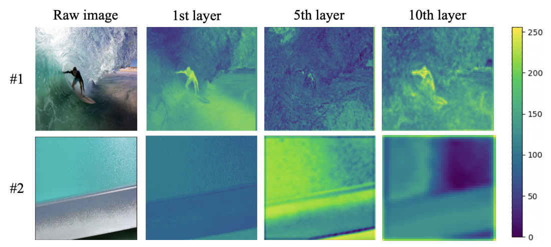

**Figure 8: Visualization of the cluster indices generated by ConvReLU++ in each Conv-ReLU layer of ResNet50. Image #1 comes from ImageNet and #2 is from Industrial Images.**

**Bound Calculation Effectiveness.** Since the upper-bound calculation, $\phi$ is a crucial component in our approach (Equation 5), we evaluate it separately by checking whether it yields a tighter bound than the normal upper-bound $\phi_{base}$ (Equation 7). As shown in Figure 9, the mean value of $\phi/\phi_{base}$ among all convolution operations in ResNet50 is 0.95, and the minimum is 0.17. Based on the distribution, our method gets a tighter bound than $\phi_{base}$ in most cases and thus can skip more vector multiplication operations. The right side of Figure 9 compares the end-to-end FLOPs reduction obtained by using ConvReLU++ with different bound estimation methods. Our upper-bound estimation method leads to about 1.5% higher computation reduction than the normal bound calculation method.

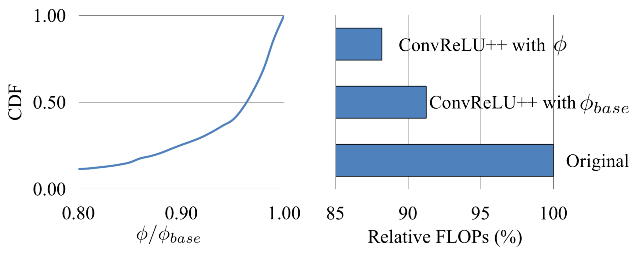

**Figure 9: The distribution of $\frac{\phi}{\phi_{base}}$ ratios in all convolution operations (left) and relative FLOPs achieved by our method with different bound calculation methods (right), obtained with ResNet50 on Industrial Images.**

## 6.4 Latency Reduction

We further examine whether the FLOPs reduction of ConvReLU++ can map to actual latency reduction on real edge devices. Specifically, we test five inference tasks on two representative devices including an Android smartphone and an Arduino Nano board. As shown in Table 4, using ConvReLU++ can lead to latency reduction on both the smartphone and the Arduino board, demonstrating the applicability of ConvReLU++ on real-world edge devices.

However, we also observe that the latency reduction is smaller than the FLOPs reduction (*i.e.,* the theoretical upper bound of latency reduction). There are several reasons for this. First, precomputing and retrieving the reference inputs and outputs during the Conv-ReLU operation may break the sequential pattern of memory access, which leads to a higher cache miss rate and slows down the computation. Second, the inference framework runs with multi-threading parallelism on the smartphone, the skippable vector multiplications in ConvReLU++ are unable to fully utilize the parallelism as dense matrix multiplication. Third, the model used on Arduino Nano is int8-quantized, but the hash function in our ConvReLU++ kernel is based on the float version model. Some of the above issues can be mitigated with engineering efforts, which we leave for future work. Nevertheless, the latency reduction is at a similar level as FLOPs reduction, which demonstrates the compatibility of ConvReLU++ with mobile platforms.

## 6.5 System Memory Overhead

ConvReLU++ does not have a time overhead since it requires no training or preparation. The overhead of ConvReLU++ is mainly on the memory because it requires extra space to maintain the pre-computed references. We measure the memory taken by the inference frameworks on the devices. As shown in Table 4, the memory overhead of ConvReLU++ is almost negligible (less than 2.60%). This is a reasonable result since ConvReLU++ only needs to retain a small number of input patches (*i.e.,* the cluster centroids) in the memory, which is minimal as compared with the memory taken by the model and feature maps.

**Table 4: Latency reduction and memory overhead of ConvReLU++ on real edge devices.**

| Device | Model + Dataset | Original Lat. | Our Lat. | Reduced FLOPs | Original Mem. | Our Mem. |
|---|---|---|---|---|---|---|
| Smartphone | ResNet50+Industrial Images | 4.86s | 4.47s (-8.02%) | -11.81% | 148.40M | 152.30M (+2.6%) |
| Smartphone | MobileNet-SSD+TSRD | 1.08s | 1.01s (-6.44%) | -10.82% | 83.10M | 84.20M (+1.32%) |
| Smartphone | SqueezeNet+ImageNet | 0.24s | 0.23s (-2.90%) | -5.28% | 43.50M | 44.30M (+1.84%) |
| Arduino | MobileNet+PnPLO | 1.46s | 1.33s (-8.91%) | -9.54% | 198.23K | 200.61K (+1.20%) |
| Arduino | MobileNet+COCO | 1.46s | 1.36s (-7.16%) | -8.37% | 198.23K | 200.61K (+1.20%) |

## 7 DISCUSSION

**Implementation on accelerators.** While we prototype the inference stage of ConvReLU++ on CPU, we expect that it can be ported to and benefit from hardware accelerators. Taking GPU as an example, our method is capable of improving the inference speed by reducing the number of GPU threads needed for computing output feature maps.

**Compatibility with other optimizations.** Modern inference framework is very complicated with various other optimization techniques (packing, Winograd, etc.) [23, 34]. Our current implementation does not consider these optimizations, but we expect supporting them to be feasible since our kernel is parallelizable by design.

**Further acceleration with training.** Since ConvReLU++ mainly benefits from the similarity between input patches, it exists an opportunity to increase the similarity through training, which can help to skip more negative-output vector multiplications.

**Limitations.** Firstly, the proposed acceleration method has different effects on multi-modal data, where continuous data with strong continuity properties in the natural world, such as images, will have better acceleration effects, while the acceleration effect of text-based data is not as significant. Secondly, the acceleration effect is more evident for shallower convolution layers due to the spatial locality properties of the shallow features, which limits the applicability of the method to deeper networks. The method also does not support other activation functions like sigmoid. Finally, the proposed conv-relu kernel may lose its acceleration effect in certain scenarios, which needs to be further investigated. These limitations highlight the need for further research to develop more effective acceleration methods that apply to a wider range of network structures and data types.

## 8 RELATED WORK

Recent work on exploiting negative-outputs detections for CNN acceleration can coarsely be divided into two groups, including approximate methods and rigorous methods.

**Approximate negative-outputs detection for single image CNN inference.** Before our work, many approaches are proposed to utilize the output sparsity of Conv-ReLU structures. For example, SparseNN [56] uses a low-rank approximation of original matrix multiplications as the negative-output detector. SeerNet [6] quantizes the original parameters, conducts low-bit multiplications, and uses the low-bit outputs for negative-output detection.

USPE [43], PredictiveNet [30] and ComPreEND [24] propose splitting values statistically based on significance for early negative detection. LCCL [14] utilizes collaborative layers to early detect and skip negative outputs calculations. However, the reduced computations in these approaches are not completely unnecessary, which may influence the prediction results.

**Approximate negative-outputs detection for continuous video CNN inference.** In video analysis tasks, there are naturally more opportunities to skip unnecessary computations due to the temporal locality. For example, Skip-Conv [16] proposes to skip negative residual areas in video frames and only update value within non-negative residual areas. DeepCache [53], Recurrent Residual Module (RRM) [37] and CBinfer [7] cache input and output feature maps from previous frames at every convolutional layer to process the differences, increase sparsity and then accumulate the outputs together with dense outputs from previous frames. DeltaCNN [38] performs all operations sparsely, by comparing only the input features (camera image) against the complete previous input, propagating the sparse feature updates throughout all layers.

**Rigorous negative-outputs detection for continuous video CNN inference.** Wakatsuki et al. [46] propose the first lossless acceleration method that only leverages temporal locality to identify negative-outputs areas for continuous vision inference. Our method can losslessly accelerate general vision tasks (both single image and continuous video) via rigorous negative-outputs detection without failures. Many other system optimization techniques that do not change the model structures or operators are also able to achieve lossless acceleration. Our work is orthogonal with them.

## 9 CONCLUSION

In this paper, we propose to losslessly skip vector multiplications in Conv-ReLU layers via foreseen output sparsity. Specifically, ConvReLU++ utilizes pre-computed references to identify negative-output vector multiplications that can be skipped. The main techniques in our approach include a hash-based reference selection mechanism, a tight upper-bound calculation method, and an end-to-end implementation on mobile platforms. Experiments have shown that ConvReLU++ achieves 2.62% to 43.77% computation reduction on various common deep vision tasks. We have also implemented the method on popular mobile inference frameworks including *ncnn* and TFLM, which can be used by edge AI application developers transparently without modifying the model.

## ACKNOWLEDGMENTS

This work is supported by the National Key R&D Program of China (2022YFF0604501) and the National Natural Science Foundation of China (62272261). We thank the anonymous reviewers and shepherds for their valuable suggestions.

## REFERENCES

[1] Vahideh Akhlaghi, Amir Yazdanbakhsh, Kambiz Samadi, Rajesh K. Gupta, and Hadi Esmaeilzadeh. 2018. SnaPEA: Predictive Early Activation for Reducing Computation in Deep Convolutional Neural Networks. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 662–673. https://doi.org/10.1109/ISCA.2018.00061

[2] Cesare Alippi, Simone Disabato, and Manuel Roveri. 2018. Moving Convolutional Neural Networks to Embedded Systems: The AlexNet and VGG-16 Case. In *2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. 212–223. https://doi.org/10.1109/IPSN.2018.00049

[3] Maria Baldeon-Calisto and Susana K. Lai-Yuen. 2020. AdaResU-Net: Multiobjective adaptive convolutional neural network for medical image segmentation. *Neurocomputing* 392 (2020), 325–340. https://doi.org/10.1016/j.neucom.2019.01.110

[4] Ali Borji, Simone Frintrop, Dicky N. Sihite, and Laurent Itti. 2012. Adaptive object tracking by learning background context. In *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. 23–30. https://doi.org/10.1109/CVPRW.2012.6239191

[5] Alessio Burrello, Angelo Garofalo, Nazareno Bruschi, Giuseppe Tagliavini, Davide Rossi, and Francesco Conti. 2021. DORY: Automatic End-to-End Deployment of Real-World DNNs on Low-Cost IoT MCUs. *IEEE Trans. Comput.* 70, 8 (2021), 1253–1268. https://doi.org/10.1109/TC.2021.3066883

[6] Shijie Cao, Lingxiao Ma, Wencong Xiao, Chen Zhang, Yunxin Liu, Lintao Zhang, Lanshun Nie, and Zhi Yang. 2019. SeerNet: Predicting Convolutional Neural Network Feature-Map Sparsity Through Low-Bit Quantization. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 11208–11217. https://doi.org/10.1109/CVPR.2019.01147

[7] Lukas Cavigelli and Luca Benini. 2020. CBinfer: Exploiting Frame-to-Frame Locality for Faster Convolutional Network Inference on Video Streams. *IEEE Transactions on Circuits and Systems for Video Technology* 30, 5 (2020), 1451–1465. https://doi.org/10.1109/TCSVT.2019.2903421

[8] Ziqian Chen, Shiqi Wang, Dapeng Oliver Wu, Tiejun Huang, and Ling-Yu Duan. 2018. From Data to Knowledge: Deep Learning Model Compression, Transmission and Communication. In *Proceedings of the 26th ACM International Conference on Multimedia* (Seoul, Republic of Korea) *(MM '18)*. Association for Computing Machinery, New York, NY, USA, 1625–1633. https://doi.org/10.1145/3240508.3240654

[9] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. cuDNN: Efficient Primitives for Deep Learning. *CoRR* abs/1410.0759 (2014). arXiv:1410.0759 http://arxiv.org/abs/1410.0759

[10] L. Dagum and R. Menon. 1998. OpenMP: an industry standard API for shared-memory programming. *IEEE Computational Science and Engineering* 5, 1 (1998), 46–55. https://doi.org/10.1109/99.660313

[11] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, Pete Warden, and Rocky Rhodes. 2021. TensorFlow Lite Micro: Embedded Machine Learning for TinyML Systems. In *Proceedings of Machine Learning and Systems*, Vol. 3. 800–811.

[12] 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*. 248–255. https://doi.org/10.1109/CVPR.2009.5206848

[13] Simone Disabato, Manuel Roveri, and Cesare Alippi. 2021. Distributed Deep Convolutional Neural Networks for the Internet-of-Things. *IEEE Trans. Comput.* 70, 8 (2021), 1239–1252. https://doi.org/10.1109/TC.2021.3062227

[14] Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. 2017. More is Less: A More Complicated Network with Less Inference Complexity. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1895–1903. https://doi.org/10.1109/CVPR.2017.205

[15] 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* (Providence, RI, USA) *(ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 199–213. https://doi.org/10.1145/3297858.3304011

[16] Amirhossein Habibian, Davide Abati, Taco S. Cohen, and Babak Ehteshami Bejnordi. 2021. Skip-Convolutions for Efficient Video Processing. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2694–2703. https://doi.org/10.1109/CVPR46437.2021.00272

[17] John A Hartigan and Manchek A Wong. 1979. Algorithm AS 136: A K-means Clustering Algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28 (1979), 100–108.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. https://doi.org/10.1109/CVPR.2016.90

[19] Changwan Hong, Aravind Sukumaran-Rajam, Israt Nisa, Kunal Singh, and P. Sadayappan. 2019. Adaptive Sparse Tiling for Sparse Matrix Multiplication. In *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*.

[20] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR* abs/1704.04861 (2017). arXiv:1704.04861 http://arxiv.org/abs/1704.04861

[21] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size. *CoRR* abs/1602.07360 (2016). arXiv:1602.07360 http://arxiv.org/abs/1602.07360

[22] Shiqi Jiang, Zhiqi Lin, Yuanchun Li, Yuanchao Shu, and Yunxin Liu. 2021. Flexible high-resolution object detection on edge devices with tunable latency. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*. 559–572.

[23] Xiaotang Jiang, Huan Wang, Yiliu Chen, Ziqi Wu, Lichuan Wang, Bin Zou, Yafeng Yang, Zongyang Cui, Yu Cai, Tianhang Yu, Chengfei Lyu, and Zhihua Wu. 2020. MNN: A Universal and Efficient Inference Engine. In *Proceedings of Machine Learning and Systems*, Vol. 2. 1–13.

[24] Namhyung Kim, Hanmin Park, Dongwoo Lee, Sungbum Kang, Jinho Lee, and Kiyoung Choi. 2022. ComPreEND: Computation Pruning through Predictive Early Negative Detection for ReLU in a Deep Neural Network Accelerator. *IEEE Trans. Comput.* 71, 7 (2022), 1537–1550. https://doi.org/10.1109/TC.2021.3092205

[25] Hugo Larochelle, Dumitru Erhan, Aaron C. Courville, James Bergstra, and Yoshua Bengio. 2007. An empirical evaluation of deep architectures on problems with many factors of variation. In *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvalis, Oregon, USA, June 20-24, 2007 (ACM International Conference Proceeding Series, Vol. 227)*. ACM, 473–480. https://doi.org/10.1145/1273496.1273556

[26] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324. https://doi.org/10.1109/5.726791

[27] Seulki Lee and Shahriar Nirjon. 2020. Learning in the Wild: When, How, and What to Learn for On-Device Dataset Adaptation. In *Proceedings of the 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things* (Virtual Event, Japan) *(AIChallengeIoT '20)*. Association for Computing Machinery, New York, NY, USA, 34–40. https://doi.org/10.1145/3417313.3429382

[28] Yanhong Li, D. Lopresti, G. Nagy, and A. Tomkins. 1996. Validation of image defect models for optical character recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 18, 2 (1996), 99–107. https://doi.org/10.1109/34.481536

[29] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common Objects in Context. In *Computer Vision – ECCV 2014*. Springer International Publishing, Cham, 740–755.

[30] Yingyan Lin, Charbel Sakr, Yongjune Kim, and Naresh Shanbhag. 2017. PredictiveNet: An energy-efficient convolutional neural network via zero prediction. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–4. https://doi.org/10.1109/ISCAS.2017.8050797

[31] Bingyan Liu, Yuanchun Li, Yunxin Liu, Yao Guo, and Xiangqun Chen. 2020. Pmc: A privacy-preserving deep learning model customization framework for edge computing. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 4 (2020), 1–25.

[32] Shuying Liu and Weihong Deng. 2015. Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*. 730–734. https://doi.org/10.1109/ACPR.2015.7486599

[33] Saeeda Naz, Khizar Hayat, Muhammad Imran Razzak, Muhammad Waqas Anwar, Sajjad A. Madani, and Samee U. Khan. 2014. The Optical Character Recognition of Urdu-like Cursive Scripts. *Pattern Recogn.* 47, 3 (mar 2014), 1229–1248. https://doi.org/10.1016/j.patcog.2013.09.037

[34] Nihui. 2018. NCNN is a high-performance neural network inference framework optimized for the mobile platform. http://github.com/tencent/ncnn.

[35] Lin Ning and Xipeng Shen. 2019. Deep Reuse: Streamline CNN Inference on the Fly via Coarse-Grained Computation Reuse. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3330345.3330384

[36] NVIDIA. 2022. cuBLAS Library. https://developer.nvidia.com/cublas

[37] Bowen Pan, Wuwei Lin, Xiaolin Fang, Chaoqin Huang, Bolei Zhou, and Cewu Lu. 2018. Recurrent Residual Module for Fast Inference in Videos. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1536–1545. https://doi.org/10.1109/CVPR.2018.00166

[38] Mathias Parger, Chengcheng Tang, Christopher D. Twigg, Cem Keskin, Robert Wang, and Markus Steinberger. 2022. DeltaCNN: End-to-End CNN Inference of Sparse Frame Differences in Videos. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 12487–12496. https://doi.org/10.1109/CVPR52688.2022.01217

[39] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1* (Montreal, Canada) *(NIPS'15)*. MIT Press, Cambridge, MA,

USA, 91–99.

[40] Francisco Romero, Qian Li 0027, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2021. INFaaS: Automated Model-less Inference Serving. In *2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021*. USENIX Association, 397–411. https://www.usenix.org/conference/atc21/presentation/romero

[41] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Springer International Publishing, Cham, 234–241.

[42] Lili Song, Ying Wang, Yinhe Han, Xin Zhao, Bosheng Liu, and Xiaowei Li. 2016. C-Brain: A Deep Learning Accelerator That Tames the Diversity of CNNs through Adaptive Data-Level Parallelization. In *Proceedings of the 53rd Annual Design Automation Conference* (Austin, Texas) *(DAC'16)*. Association for Computing Machinery, New York, NY, USA, Article 123, 6 pages. https://doi.org/10.1145/2897937.2897995

[43] Mingcong Song, Jiechen Zhao, Yang Hu, Jiaqi Zhang, and Tao Li. 2018. Prediction Based Execution on Deep Neural Networks. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 752–763. https://doi.org/10.1109/ISCA.2018.00068

[44] Ke Tan and DeLiang Wang. 2021. Towards Model Compression for Deep Learning Based Speech Enhancement. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 29 (2021), 1785–1794. https://doi.org/10.1109/TASLP.2021.3082282

[45] TIANCHI-Alibaba. 2018. Industrial Defect Dataset. https://tianchi.aliyun.com/competition/entrance/231682/introduction?lang=en-us

[46] Toshiaki Wakatsuki, Sekitoshi Kanai, and Yasuhiro Fujiwara. 2021. Accelerate Inference of CNNs for Video Analysis While Preserving Exactness Exploiting Activation Sparsity. In *Proceedings of Machine Learning and Systems*, Vol. 3. 860–872.

[47] Chao Wang, Lei Gong, Qi Yu, Xi Li, Yuan Xie, and Xuehai Zhou. 2017. DLAU: A Scalable Deep Learning Accelerator Unit on FPGA. *Trans. Comp.-Aided Des. Integ. Cir. Sys.* 36, 3 (mar 2017), 513–517. https://doi.org/10.1109/TCAD.2016.2587683

[48] Qiang Wang, Li Zhang, Luca Bertinetto, Weiming Hu, and Philip H.S. Torr. 2019. Fast Online Object Tracking and Segmentation: A Unifying Approach. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 1328–1338. https://doi.org/10.1109/CVPR.2019.00142

[49] Guojun Wen, Zhijun Gao, Qi Cai, Yudan Wang, and Shuang Mei. 2020. A Novel Method Based on Deep Convolutional Neural Networks for Wafer Semiconductor Surface Defect Inspection. *IEEE Transactions on Instrumentation and Measurement* 69, 12 (2020), 9668–9680. https://doi.org/10.1109/TIM.2020.3007292

[50] Hao Wen, Yuanchun Li, Zunshuai Zhang, Shiqi Jiang, Xiaozhou Ye, Ye Ouyang, Ya-Qin Zhang, and Yunxin Liu. 2023. AdaptiveNet: Post-deployment Neural Architecture Adaptation for Diverse Edge Environments. *arXiv preprint arXiv:2303.07129* (2023).

[51] Xinxin Wu, Zhihua Fan, Tianyu Liu, Wenming Li, Xiaochun Ye, and Dongrui Fant. 2022. LRP: Predictive output activation based on SVD approach for CNN s acceleration. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 831–836. https://doi.org/10.23919/DATE54114.2022.9774744

[52] Luofeng Xie, Xiao Xiang, Huining Xu, Ling Wang, Lijun Lin, and Guofu Yin. 2021. FFCNN: A Deep Neural Network for Surface Defect Detection of Magnetic Tile. *IEEE Transactions on Industrial Electronics* 68, 4 (2021), 3506–3516. https://doi.org/10.1109/TIE.2020.2982115

[53] Mengwei Xu, Mengze Zhu, Yunxin Liu, Felix Xiaozhu Lin, and Xuanzhe Liu. 2018. DeepCache: Principled Cache for Mobile Deep Vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking* (New Delhi, India) *(MobiCom '18)*. Association for Computing Machinery, New York, NY, USA, 129–144. https://doi.org/10.1145/3241539.3241563

[54] Jinrui Zhang, Huan Yang, Ju Ren, Deyu Zhang, Bangwen He, Ting Cao, Yuanchun Li, Yaoxue Zhang, and Yunxin Liu. 2022. MobiDepth: real-time depth estimation using on-device dual cameras. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*. 528–541.

[55] Minjia Zhang, Samyam Rajbhandari, Wenhan Wang, and Yuxiong He. 2018. DeepCPU: Serving RNN-based Deep Learning Models 10x Faster. In *2018 USENIX Annual Technical Conference, USENIX ATC 2018, Boston, MA, USA, July 11-13, 2018*. USENIX Association, 951–965. https://www.usenix.org/conference/atc18/presentation/zhang-minjia

[56] Jingyang Zhu, Jingbo Jiang, Xizi Chen, and Chi-Ying Tsui. 2018. SparseNN: An energy-efficient neural network accelerator exploiting input and output sparsity. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 241–244. https://doi.org/10.23919/DATE.2018.8342010

[57] Jingyang Zhu, Zhiliang Qian, and Chi-Ying Tsui. 2016. LRADNN: High-throughput and energy-efficient Deep Neural Network accelerator using Low Rank Approximation. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. 581–586. https://doi.org/10.1109/ASPDAC.2016.7428074

[58] Zhe Zhu, Dun Liang, Songhai Zhang, Xiaolei Huang, Baoli Li, and Shimin Hu. 2016. Traffic-Sign Detection and Classification in the Wild. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2110–2118. https://doi.org/10.1109/CVPR.2016.232