RNNPool: Efficient Non-linear Pooling for RAM Constrained Inference

Oindrila Saha[†], **Aditya Kusupati**[‡],
Harsha Vardhan Simhadri[†], **Manik Varma**[†] **and Prateek Jain**[†]
[†]Microsoft Research India, [‡]University of Washington
{t-oisaha,harshasi,manik,prajain}@microsoft.com, kusupati@cs.washington.edu

Abstract

Standard Convolutional Neural Networks (CNNs) designed for computer vision tasks tend to have large intermediate activation maps. These require large working memory and are thus unsuitable for deployment on resource-constrained devices typically used for inference on the edge. Aggressively downsampling the images via pooling or strided convolutions can address the problem but leads to a significant decrease in accuracy due to gross aggregation of the feature map by standard pooling operators. In this paper, we introduce RNNPool, a novel pooling operator based on Recurrent Neural Networks (RNNs), that efficiently aggregates features over large patches of an image and rapidly downsamples activation maps. Empirical evaluation indicates that an RNNPool layer can effectively replace multiple blocks in a variety of architectures such as MobileNets, DenseNet when applied to standard vision tasks like image classification and face detection. That is, RNNPool can significantly decrease computational complexity and peak memory usage for inference while retaining comparable accuracy. We use RNNPool with the standard S3FD [50] architecture to construct a face detection method that achieves state-of-the-art MAP for tiny ARM Cortex-M4 class microcontrollers with under 256 KB of RAM. Code is released at https://github.com/Microsoft/EdgeML.

1 Introduction

Convolutional Neural Networks (CNNs) have become ubiquitous for computer vision tasks such as image classification and face detection. Steady progress has led to new CNN architectures that are increasingly accurate, but also require larger memory and more computation for inference. The increased inference complexity renders these models unsuitable for resource-constrained processors that are commonplace on the edge in IoT systems and battery-powered and privacy-centric devices.

To reduce inference complexity, several techniques like quantization [44], sparsification [9, 27], cheaper CNN blocks [37, 22], or neural architecture search [41] have been proposed to train CNN models with lower inference cost and model size while retaining accuracy. However, these models still require large working memory for inference. Memory tends to be the most constrained resource on low power devices as it occupies a large fraction of the device die and has high sustained power requirement [24]. Most low power ARM Cortex-M* microcontrollers have less than 256 KB RAM.

Typical CNNs have large intermediate activation maps, as well as many convolution layers, which put together require large amount of RAM for inference (see Proposition 1). A standard approach to reducing working memory is to use pooling operators or strided convolution to bring down size of the activation map. In fact, standard CNNs have multiple such layers. However, such pooling operators aggregate the underlying activation map in a simplistic manner, which can lead to a significant loss of accuracy. As a result, their use is limited to small receptive fields, typically no larger than 3×3, and they can not be used to aggressively reduce the activation map by aggregating larger receptive fields.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

In this paper, we propose a novel pooling operator RNNPool that uses Recurrent Neural Networks (RNNs) to perform a more refined aggregation over a large receptive field of the activation map without compromising on accuracy. RNNPool can be applied to any tensor structured problem, but we focus on 2D images for ease of exposition. For images, RNNPool uses RNNs to aggregate information along rows & columns in a given patch. RNNPool has three parameters – patch size or receptive field, stride, and output dimension – to control its expressiveness and ability to downsample. The RNNPool operator matches standard pooling operators syntactically, so can be used to replace them in convolutional networks.

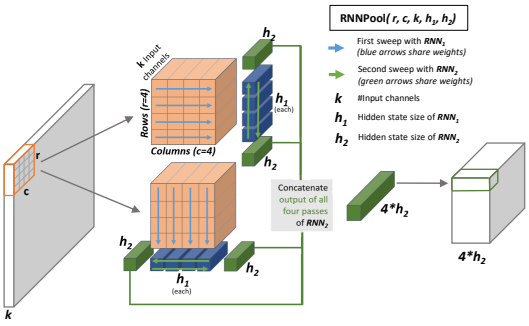

Figure 1: The RNNPool operator applied to patches of size $r \times c$ with stride s. It summarizes the patch with two RNNs into a vector of size $4h_2$.

RNNPool allows rapid down-sampling of images and activation maps, eliminating the need for many memory-intensive intermediate layers. RNNPool is most effective when used to replace multiple CNN blocks in the initial stages of the network where the activation map sizes are large, and hence, require the most memory and compute. There, a single layer of RNNPool can down-sample by a factor of 4 or 8. For example, RNNPool applied to a $640 \times 640 \times 3$ image with patch-size 16, stride 8, and 32 output channels results in a $80 \times 80 \times 32$ activation map, which can be stored in about 200 KB, and can be computed one patch at a time without significant memory cost. Replacing a few blocks using RNNPool reduces peak memory requirement significantly for typical CNN architectures without much loss of accuracy.

Our experiments demonstrate that RNNPool can be used as an effective replacement for multi-layered, expensive CNN blocks in a variety of architectures such as MobileNets, DenseNets, S3FD, and for varied tasks such as image classification and face detection. For example, in a 10-class image classification task, RNNPool+MobileNetV2 reduces the peak memory requirement of MobileNetV2 by up to $10\times$ and MAdds (MAdds refers to Multiply-Adds as in MobileNetV2 [37]) by about 25%, while maintaining the *same* accuracy. Additionally, due to its general formulation, RNNPool can replace pooling layers anywhere in the architecture. For example, it can replace the final average pool layer in MobileNetV2 and improve accuracy by $\sim 1\%$.

Finally, we modify the S3FD [50] architecture with RNNPool to construct an accurate face detection model which needs only 225 KB RAM – small enough to be deployed on a Cortex-M4 based device – and achieves 0.78 MAP on the medium category of the WIDER FACE dataset [47] using $80\times$ fewer MAdds than EXTD [48] – a state-of-the-art resource-constrained face detection method.

In summary, we make the following contributions:

- A novel pooling operator that can rapidly down-sample input in a variety of standard CNN architectures, e.g., MobileNetV2, DenseNet121 while retaining the expressiveness.
- Demonstrate that RNNPool can reduce working memory and compute requirements for image classification and Visual Wake Words significantly while retaining comparable accuracy.
- By combining RNNPool with S3FD, we obtain a state-of-the-art face detection model for ARM Cortex-M4 class devices.

2 Related Work

Pooling: Max-pooling, average-pooling and strided convolution layers [29] are standard techniques for feature aggregation and for reducing spatial resolution in CNNs. Existing literature on rethinking pooling [51, 15, 10] focuses mainly on increasing accuracy and does not take compute/memory efficiency into consideration which is the primary focus of this paper.

Efficient CNN architectures: Most existing research on efficient CNN architectures aims at reducing model size and number of operations per inference. These methods include designing new architectures such as DenseNet [21], MobileNets [20, 37] or searching for them (ProxylessNAS [3], EfficientNets [41], SqueezeNAS [38]). These architectures do not primarily optimize for the peak working memory, which is a critical constraint on devices powered by tiny microcontrollers. Previ-

ous work on memory-optimized inference manipulates existing convolution operator by reordering computations [5, 28] or performing them in place [13]. However, most of these methods provide relatively small memory savings and are validated on low-resolution images like CIFAR-10 [25]. Channel pruning [17] is a method that tries to reduce memory requirement by pruning out multiple convolution kernels in every layer. While effective, channel/filter pruning does not tackle gradual spatial downsampling and thus is a complementary technique to RNNPool.

Visual Wake Words: Visual cues (visual wake word) to "wake-up" AI-powered home assistant devices require real-time inference on relatively small devices. Chowdhery et al. [6] proposed a Visual Wake Words dataset and a resource-constrained setting to evaluate various methods. Section 5.2 discusses the efficient RNNPool based models and their performance for this task.

Face-detection on tiny devices: Recent work including EXTD [48], LFFD [18], FaceBoxes [49] and EagleEye [52] address the problem of accurate real-time face detection on resource-constrained devices. EXTD and LFFD are the most accurate but have high compute and memory requirements. On the other hand, EagleEye and FaceBoxes have lower inference complexity but also suffer from lower MAP scores. Face detection using RNNPool is discussed in Section 5.3.

RNNs for Computer Vision: RNNs have been successful for sequential tasks but haven't been extensively explored in the context of computer vision. An early work, ReNet [42], uses RNN based layer as a replacement for a convolution layer but does not aim at improving efficiency. RNNPool contrasts with ReNet as follows:

a. ReNet is designed to replace a convolutional layer by capturing the global context and leaves the local context to be captured by flattening non-overlapping patches. RNNPool, on the other hand, uses overlapping patches and strongly captures local features and relies on subsequent standard convolutions to capture the global context. Hence, RNNPool and ReNet are complementary methods and can be combined.
b. Semantically, RNNPool is a generalized pooling operator and can replace any pooling layer or strided convolution. However, ReNet does not correspond to any pooling abstraction, making it hard to combine with existing CNN models. For example, RNNPool can modify S3FD architecture to achieve state-of-the-art real-time face detection with < 1 MB RAM while ReNet fails to fit in that context as a replacement layer since the receptive field of the output of ReNet layer varies across spatial positions.
c. ReNet can still be used as a rapid downsampling layer. Table 2 shows that RNNPool outperforms ReNet with lower model size and fewer MAdds across datasets and architectures. E.g. ReNet+MobileNetV2 applied to ImageNet-1K is almost 4% less accurate than RNNPool +MobileNetV2, despite the same working RAM requirement and more MAdds per inference.

Inside-Outside Net [2] uses a ReNet based layer for extracting context features in object detection while PiCANet [31] uses it as a global attention function for salient object detection. L-RNN [45] inserts multiple ReNet based layers but in a cascading fashion. See Appendix B for more discussion.

PolygonRNN [1], CNN-RNN [43] and Conv-LSTM [46] also use RNNs in their architectures but only to model certain sequences in the respective tasks rather than tackling pooling and efficiency.

3 What is RNNPool?

Consider the output of an intermediate layer in a CNN of size $R \times C \times f$, where R and C are the number of rows and columns and f is the number of channels. A typical 2×2 pooling layer (e.g. max or average) with stride 2 would halve the number of rows and columns. So, reducing dimensions by a factor of 4 would require *two* such blocks of convolutions and pooling. Our goal is to reduce the activation of size $R \times C \times f$ to, say, $R/4 \times C/4 \times f'$ or smaller in a single layer while retaining the information necessary for the downstream task. We do so using an RNNPoolLayer illustrated in Figure 1 that utilizes strided RNNPool operators.

3.1 The RNNPool Operator and the RNNPoolLayer

An RNNPool operator of size (r, c, k, h_1, h_2) takes as input an activation patch of size $r \times c \times k$ corresponding to k input channels, and uses a pair of RNNs – RNN_1 of hidden dimension h_1 and RNN_2 with hidden dimension h_2 – to sweep the patch horizontally and vertically to produce a summary of size $1 \times 1 \times 4h_2$.

Algorithm 1 describes the RNNPool operator wich applies two parallel pipelines to a patch and concatenates their outputs. In the first, RNN_1 traverses each row and summarizes the patch horizontally (Line 12) and then RNN_2 trverses the outputs of RNN_1 (Lines 13-14) bi-directionally. In the second pipeline RNN_1 first traverses along columns to summarize the patch vertically (Line 15) and then RNN_2 (Lines 16-17) summarizes bi-directionally.

While it is possible to use GRU [4] or LSTM [19] for the two instances of RNN in RNNPool, we use FastGRNN [26] for its compact size and fewer MAdds (see Appendix H).

An RNNPoolLayer consists of a single RNNPool operator strided over an input activation map and takes as input two more parameters: patch size and the stride length. Note that there are only two RNNs (RNN_1 & RNN_2) in an RNNPool operator, thus weights are shared for both the row-wise and column-wise passes (RNN_1) and all bi-directional passes (RNN_2) across every instance of RNNPool in an RNNPoolLayer.

Algorithm 1 RNNPool Operation

Input: $\mathbf{X} : [\mathbf{x}_{1,1} \dots \mathbf{x}_{r,c}]; \mathbf{x}_{i,j} \in \mathcal{R}^k$
Output: $\text{RNNPool}(\mathbf{X})$

1: **function** $\text{FastGRNN}(\mathcal{P}, \mathbf{x})$
2: $\quad[\mathbf{W}, \mathbf{U}, \mathbf{b}_z, \mathbf{b}_h] \leftarrow \mathcal{P}$, $\mathbf{h}_0 \leftarrow \text{randn}$
3: $\quad$**for** $k \leftarrow 1$ to $\text{length}(\mathbf{x})$ **do**
4: $\quad\quad\mathbf{z} \leftarrow \sigma(\mathbf{W}\mathbf{x}_k + \mathbf{U}\mathbf{h}_{k-1} + \mathbf{b}_z)$
5: $\quad\quad\tilde{\mathbf{h}}_k \leftarrow \tanh(\mathbf{W}\mathbf{x}_k + \mathbf{U}\mathbf{h}_{k-1} + \mathbf{b}_h)$
6: $\quad\quad\mathbf{h}_k \leftarrow \mathbf{z} \odot \mathbf{h}_{k-1} + (\mathbf{1} - \mathbf{z}) \odot \tilde{\mathbf{h}}_k$
7: $\quad$**end for**
8: $\quad$**return** $\mathbf{h}_T$
9: **end function**

10: $\text{RNN}_i(\_) \leftarrow \text{FastGRNN}(\mathcal{P}_i, \_)$, for $i \in \{1, 2\}$
11: **function** $\text{RNNPool}(\mathbf{X})$
12: $\quad\mathbf{p}_i^r \leftarrow \text{RNN}_1(\mathbf{X}_{i,1 \le j \le c})$, for all $1 \le i \le r$
13: $\quad\mathbf{q}^{r_1} \leftarrow \text{RNN}_2(\mathbf{p}^r_{1 \le i \le r})$
14: $\quad\tilde{\mathbf{p}}^r \leftarrow \text{reverse}(\mathbf{p}^r)$, $\mathbf{q}^{r_2} \leftarrow \text{RNN}_2(\tilde{\mathbf{p}}^r_{1 \le i \le r})$
15: $\quad\mathbf{p}_j^c \leftarrow \text{RNN}_1(\mathbf{X}_{1 \le i \le r, j})$, for all $1 \le j \le c$
16: $\quad\mathbf{q}^{c_1} \leftarrow \text{RNN}_2(\mathbf{p}^c_{1 \le j \le c})$
17: $\quad\tilde{\mathbf{p}}^c \leftarrow \text{reverse}(\mathbf{p}^c)$, $\mathbf{q}^{c_2} \leftarrow \text{RNN}_2(\tilde{\mathbf{p}}^c_{1 \le j \le c})$
18: $\quad$**return** $[\mathbf{q}^{r_1}, \mathbf{q}^{r_2}, \mathbf{q}^{c_1}, \mathbf{q}^{c_2}]$
19: **end function**

3.2 Probing the Efficacy of RNNPool

Capturing edges, orientations, and shapes: To demonstrate the capabilities of RNNs as spatial operators for vision tasks such as capturing edges, orientations, and shapes, we performed experiments on synthetic data. RNNPool learns how to capture edges, orientations, and shapes as effectively as convolutional layers which reinforces the choice of RNNs as spatial operators. Appendix C.1 provides further details of these experiments.

Comparing performance with pooling operators: We also performed experiments to contrast the down-sampling power of RNNPool against standard pooling operators on CIFAR-10 [25]. As discussed in Appendix C.2, RNNPool significantly outperforms standard pooling operators in terms of accuracy.

4 How to use the RNNPoolLayer?

RNNPool can be used to modify standard CNN architectures and reduce their working memory as well as computational requirements. Typically, such modifications involve replacing one or more stacks of convolutions and pooling layers of the "base" (original) architecture with an RNNPoolLayer and retraining from scratch. We describe architecture modification strategies here and demonstrate their effectiveness through extensive experimentation in Section 5.

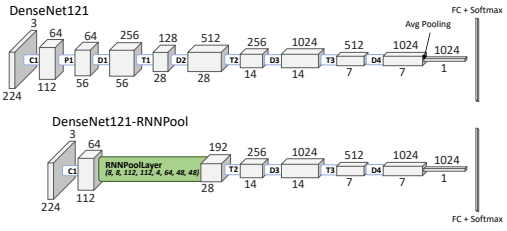

Figure 2: **DenseNet121-**RNNPool: obtained by replacing P1, D1, T1 and D2 blocks in DenseNet121 with an RNNPoolLayer.

Replacement for a Sequence of Blocks: Consider the DenseNet121 [21] architecture in Figure 2. It consists of one convolutional layer, followed by repetitions of "Dense" (D), "Transition" (T) and "Pooling" (P) blocks which gradually reduce the size of the image while increasing the number of channels. Of all these layers, the first block following the initial convolutional layer (D1) requires the most working memory and compute as it works on large activation maps that are yet to be down-sampled. Further, the presence of 6 layers within each dense block makes it harder to work with small memory (see Proposition 1). This is also true of other architectures such as MobileNetV2, EfficientNet, and ResNet.

Table 1: Comparison of inference complexity and accuracy with and without RNNPoolLayer on ImageNet-10.

Model	Base						RNNPool			
	Accuracy (%)	Parameters	Memory Optimised		Standard Calculation [6, 37]					
			Peak RAM	MAdds	Peak RAM	MAdds	Accuracy (%)	Parameters	Peak RAM	MAdds
MobileNetV2	94.20	2.20M	0.38 MB	1.00G	2.29 MB	0.30G	**94.40**	**2.00M**	**0.24 MB**	**0.23G**
EfficientNet-B0	96.00	4.03M	0.40 MB	1.09G	2.29 MB	0.39G	**96.40**	**3.90M**	**0.25 MB**	**0.33G**
ResNet18	**94.80**	11.20M	0.38 MB	21.58G	3.06 MB	1.80G	94.40	**10.60M**	**0.38 MB**	**0.95G**
DenseNet121	**95.40**	6.96M	1.53 MB	24.41G	3.06 MB	2.83G	94.80	**5.60M**	**0.77 MB**	**1.04G**
GoogLeNet	**96.00**	9.96M	1.63 MB	3.32G	3.06 MB	1.57G	95.60	**9.35M**	**0.78 MB**	**0.81G**

We can use an RNNPoolLayer to rapidly down-sample the image size and bypass intermediate large spatial resolution activations. In DenseNet121, we can replace 4 blocks - P1, D1, T1, D2 - spanning *39* layers with a single RNNPoolLayer to reduce the activation map from size $112 \times 112 \times 64$ to $28 \times 28 \times 128$ (see Figure 2). The replacement RNNPoolLayer can be executed patch-by-patch without re-computation, thus reducing the need to store the entire activation map across the image. These two factors greatly reduce the working memory size as well as the number of computations. DenseNet121-RNNPool achieves an accuracy of 94.8% on ImageNet-10 (see Appendix A for dataset details) which is comparable to 95.4% of the original DenseNet121 model.

A similar replacement of functional blocks with RNNPoolLayer can be performed for MobileNetV2 as specified in Table 10 of Appendix F, and leads to a similar reduction in the size of the largest activation map while retaining accuracy. These results extend to other networks like EfficientNet, ResNet and GoogLeNet [40], where residual connection based functional blocks in the initial parts can be effectively replaced with the RNNPoolLayer with improvements in working memory and compute, while retaining comparable accuracy. These results are listed in Table 1. Appendix H presents further ablation studies on RNNPool and its base model.

Replacement for Pooling Layers: RNNPool has the same input and output interface as any pooling operator and hence, RNNPoolLayer can replace any standard pooling layer while providing more accurate aggregation. For example, DenseNet121-RNNPool has three pooling layers one each in T2, T3, and the final average pool layer. Table 1 shows that, on ImageNet-10, DensetNet121-RNNPool loses 0.6% accuracy compared to its base model. But, replacing all three remaining pooling layers in DenseNet121-RNNPool with a RNNPoolLayer results in almost the same accuracy as the *base* DenseNet121 but with about $2\times$ and $4\times$ lower compute and RAM requirement respectively. We can further drop 14 dense layers in D3 and 10 layers in D4 to bring down MAdds and RAM requirement to 0.79G MAdds and 0.43 MB, respectively, while still ensuring 94.2% accuracy.

Replacement in Face Detection models: As in the above architectures, we can use RNNPoolLayer to rapidly down-sample the image by a factor of 4×4 in the early phase of an S3FD face detector [50]. The resulting set of architectures (with different parameters) are described in Appendix F.2. For example, the RNNPool-Face-Quant model has a state-of-the-art MAP for methods that are constrained to at most 256 KB of working RAM (Table 4).

Inference memory requirements: Computing exact memory and compute requirement of a large CNN model is challenging as the execution order of activations in various layers can be re-organized to trade-off memory and compute. For example, in the *memory-optimized* column of Table 1 we present the compute usage of a variety of baseline architectures when their execution order (EO) is restricted to using no more memory than the corresponding RNNPool based architecture. That is, we identify the memory bottleneck layers in various architectures whose activation map size is almost same as that of the corresponding RNNPool-based model. We then compute every voxel of this layer by re-computing the required set of convolutions, *without* storing them. CNNs, in general, have significant compute requirement and such re-compute intensive optimizations make the architecture infeasible even for large devices, e.g. DenseNet121 requires 24.41G MAdds in this scheme (Table 1).

A standard approach is to restrict execution orders that do not require any re-computation of intermediate activation maps. A straightforward and standard EO is the one where the computation is executed *layer-by-layer* [6, 37]. The memory requirement of such a scheme would correspond to the largest activation map in the architecture, except the output of 1x1 convolution layers which can be computed on the fly. This approach mimics the memory requirement of existing platforms like TF-lite [11] and is proposed as a standard benchmark for comparing resource-constrained inference methods [6]. Following this prior convention, we list the inference complexity for various architectures under the *compute-optimized* columns in Table 1, unless the operation is easy to compute on the fly like 1x1

Table 2: Impact of various downsampling and pooling operators on the accuracy, inference complexity and the model size of three *base* architectures: MobileNetV2 and DenseNet121 for ImageNet-10 dataset, and MobileNetV2-0.35x for Visual Wake Word dataset. First block of the table represents the base network and a modified network where the last average pooling layer in the network is replaced by RNNPoolLayer. Second block represent modified networks where the image is passed through a convolution layer followed by various downsampling methods to reduce the size of image by a factor of 4×4. The last row represents the architecture from the second block with RNNPoolLayer with an additional RNNPool replacing the last layer. Peak RAM usage computed using standard convention of [6] is the same for all methods in the second block. Note that RNNPoolLayer +Last layer RNNPool has accuracy similar to the base network while other methods like ReNet are 2-3% less accurate.

Method	**ImageNet-10**						**Visual Wake Words**		
	MobileNetV2			DenseNet121			MobileNetV2-0.35×		
	Accuracy (%)	MAdds	Parameters	Accuracy (%)	MAdds	Parameters	Accuracy (%)	MAdds	Parameters
Base Network	94.20	0.300G	2.2M	**95.40**	2.83G	6.96M	90.20	53.2M	296K
Last layer RNNPool	95.00	0.334G	2.9M	**95.40**	3.05G	7.41M	**91.14**	53.4M	300K
Average Pooling	90.80	**0.200G**	**2.0M**	92.80	**0.71G**	**5.59M**	86.85	**31.9M**	**255K**
Max Pooling	92.80	**0.200G**	**2.0M**	93.40	**0.71G**	**5.59M**	86.92	**31.9M**	**255K**
Strided Convolution	93.00	0.258G	2.1M	93.80	1.33G	6.38M	88.08	39.2M	264K
ReNet	92.20	0.296G	2.3M	93.00	1.35G	6.41M	88.10	46.4M	277K
RNNPoolLayer	94.40	0.226G	**2.0M**	94.80	1.04G	5.60M	89.57	37.7M	**255K**
RNNPoolLayer + Last layer RNNPool	**95.60**	0.260G	2.7M	95.00	1.26G	6.06M	89.65	37.9M	259K

convolution or patch-by-patch computation of RNNPool. Appendix E.2 provides more details about these calculations.

The above scheme is easy to implement and allows an inference pipeline that is more modular and easy to debug and could allow faster inference on neural network accelerators [23]. But, in principle, one can design execution orders (EO) that do not re-compute any intermediate layers, but are still not required to store entire activation maps, especially the largest ones. So, a rigorous quantification of the memory requirement of a model (without any re-compute) needs to show that any valid execution order requires a certain amount of working memory at some point in its execution, and also demonstrate a valid EO with the same memory requirement as a matching upper bound. We achieve this with the following proposition, whose proof and corollaries are in Appendix D.

Proposition 1 *Consider an l-layer ($l > 1$) convolutional network with a final layer of size $m \times n$. Suppose the for each node in the output layer, the size of receptive field in intermediate layer $q \in [l-1]$ is $(2k_q+1) \times (2k_q+1), k_q > 0$ and that this layer has c_q channels and stride 1. Any serial execution order of this network that disallows re-computation requires at least $2\sum_{q=1}^{l-1} c_q k_q \times min(m-1, n-1)$ memory for nodes in the intermediate layers.*

The above proposition shows that for a CNN with receptive field k_q at the q-th layer, the memory requirement scales linearly with the height/width of the activation map and with the number of layers. As networks like MobileNetV2 or DenseNet have blocks with a significant number of convolution layers and large receptive field, this proposition implies that it is not possible to significantly reduce the memory requirement over the standard *layer-by-layer* approach. For example, our un-optimized calculations for RNNPool architectures still give us $3-4$x reduction in peak RAM usage when compared to the *minimum* RAM requirement of the corresponding base architecture (see Appendix E.1). Further, similar optimization can be applied to RNNPool based architectures, so the relative reduction in memory by RNNPool does not change significantly. The implications of the above proposition, i.e., the peak memory of various networks without re-compute is calculated in Appendix E.1.

5 Evaluation of RNNPool on Vision Tasks

We present empirical evidence that RNNPool operator is compatible with popular CNN architectures for vision tasks, and can push the envelope of compute/memory usage vs accuracy curve. Further, we show that RNNPool combined with MobileNetV2 [37] generates accurate models for Visual wake words and face detection problems that can be deployed on tiny Cortex-M4 microcontrollers. See Appendix G for more details about model training and hyperparameters used for the experiments.

5.1 RNNPool for Image Classification

We first focus on ImageNet-10, a 10 class subset of ImageNet-1K [7] where the classes correspond to the categories in CIFAR-10 [25]. We study this dataset because in several realistic tiny devices scenario, like intrusion detection, we are interested in identifying the presence/absence of a few, rather than 1000, classes of objects. The dataset is divided into 1300 images for training and 50 for validation per class. More details and rationale about the dataset can be found in the Appendix A.

Table 2 compares RNNPoolLayer against other standard pooling operators as used in MobileNetV2 and DenseNet121 base networks (see Appendix F.1 for description of the architecture). It shows that with the same memory usage, RNNPool is up to 4% more accurate than the standard pooling operators. While standard pooling operators are cheaper than RNNPool, the overall compute requirement of RNNPool based architectures is similar to pooling based architectures. Furthermore, replacing the last average pooling layer in the base network with RNNPool further increases accuracy, thus demonstrating the flexibility of RNNPoolLayer. Table 2 also contrasts RNNPool with ReNet [42] as a downsampling layer. We observe that RNNPool is a much better alternative for downsampling layers in terms of accuracy (better by up to 2%), model size, and MAdds for the same amount of working memory.

Next, we study the compatibility of RNNPool with different architectures. Table 1 shows that RNNPool based architectures maintain the accuracy of base models while significantly decreasing memory and compute requirement. See Section 4 and Appendix E for a discussion on the calculation of memory and compute requirements of different models.

Table 3: Comparison of resources and accuracy with MobileNets for ImageNet-1K.

Method	Peak RAM	Parameters	MAdds	Accuracy (%)
MobileNetV1	3.06MB	4.2M	569M	**69.52**
MobileNetV1-ReNet	0.77MB	4.2M	487M	66.90
MobileNetV1-RNNPool	**0.77MB**	**4.1M**	**417M**	69.39
MobileNetV2	2.29MB	3.4M	300M	**71.81**
MobileNetV2-ReNet	0.24MB	3.6M	296M	66.72
MobileNetV2-RNNPool	**0.24MB**	**3.2M**	**226M**	70.14

Finally, Table 3 presents results on the complete ImageNet-1K [7] dataset with MobileNetV1 and MobileNetV2 as the base architectures. ReNet and RNNPool based models are constructed in a manner similar to the models in Table 1. See Table 10 for the complete specification of the MobileNetV2+RNNPool model. MobileNetV1+RNNPool model is constructed similarly with $h_1 = h_2 = 16$. Consistent with the results on ImageNet-10, RNNPool retains almost same accuracy as the base models while decreasing memory usage significantly. Furthermore, RNNPool based models are also $3 - 4\%$ more accurate than *ReNet based models*. In this work, we focus on state-of-the-art resource-constrained models that do not require neural architecture search (NAS); we leave extension of RNNPool for NAS based architectures like EfficientNets [41] for future work.

5.2 RNNPool for Visual Wake Words

The Visual Wake Words challenge [6] presents a relevant use case for computer vision on tiny microcontrollers. It requires detecting the presence of a human in the frame with very little resources — no more than 250 KB peak RAM usage and model size, and no more than 60M MAdds/image. The existing state-of-the-art method [6] is MobileNetV2-$0.35\times$ with 8 channels for the first convolution and 320 channels for the last convolution layer. We use this as our baseline and replace convolutions with an RNNPoolLayer. After training a floating-point model with the best validation accuracy, we perform per-channel quantization to obtain 8-bit integer weights and activations.

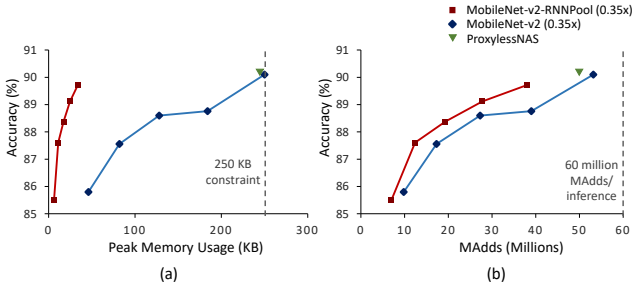

Figure 3: Visual Wake Word: MobileNetV2-RNNPool requires $8\times$ less RAM and 40% less compute than baselines. We cap the number of parameters at $\leq$ 250K instead of the 290K allowed by MobileNetV2 ($0.35\times$). ProxylessNAS has 242K parameters.

Table 2 compares the accuracy of the baseline and new architectures on this task. Replacing the last average pool layer with RNNPool increases the accuracy by $\geq 1\%$. Inserting RNNPool both at the beginning of the network and at the end provides a model whose accuracy is within 0.6% of the

Table 4: Comparison of memory requirement, no. of parameters and validation MAP of various Face Detection architectures when applied to 640×480 RGB images from the Wider Face dataset. RNNPool-Face-C achieves higher accuracy than the baselines despite using $3\times$ less RAM and $4.5\times$ less MAdds. RNNPool-Face-Quant enables deployment on Cortex-M4 class devices with 6-7% accuracy gains over the cheapest baselines.

Method	Peak RAM	Parameters	MAdds	MAP			MAP for ≤ 3 faces		
				Easy	Medium	Hard	Easy	Medium	Hard
EXTD	18.75 MB	**0.07M**	8.49G	0.90	0.88	**0.82**	0.93	0.93	0.91
LFFD	18.75 MB	2.15M	9.25G	0.91	0.88	0.77	0.83	0.83	0.82
RNNPool-Face-C	**6.44 MB**	1.52M	**1.80G**	**0.92**	**0.89**	0.70	**0.95**	**0.94**	**0.92**
FaceBoxes	1.76 MB	**1.01M**	2.84G	0.84	0.77	0.39	-	-	-
RNNPool-Face-B	1.76 MB	1.12M	**1.18G**	**0.87**	**0.84**	**0.67**	0.91	0.90	0.88
EagleEye	1.17 MB	0.23M	**0.08G**	0.74	0.70	0.44	0.79	0.78	0.75
RNNPool-Face-A	1.17 MB	**0.06M**	0.10G	0.77	0.75	0.53	0.81	0.79	0.77
RNNPool-Face-Quant	**225 KB**	0.07M	0.12G	**0.80**	**0.78**	**0.53**	**0.84**	**0.83**	**0.81**

baseline but with far smaller memory requirement (250 $\rightarrow$ 33.68 KB), model size, and MAdds. Peak memory usage is calculated using the same convention as [6].

Further, we sweep across input image resolutions of {96, 128, 160, 192, 224} to trade-off between accuracy and efficiency. Figure 3 shows that RNNPool models are significantly cheaper during inference in terms of compute and memory while offering the same accuracy as the baselines. For example, peak memory usage of MobileNetV2-0.35$\times$ with the lowest resolution images is $\sim$40 KB, while our model requires only 34 KB RAM despite using the highest resolution image and providing $\sim$4% higher accuracy. Note that ProxylessNAS [14] was the winner of the Visual Wake Words challenge. We report it's accuracy on the final network provided by the authors. To be consistent, we train the model only on the training data provided, instead of pretraining with ImageNet-1K used by ProxylessNAS in the wake word challenge.

5.3 RNNPool for Face Detection

We experiment with multiple architectures we call RNNPool-Face-* for face detection suggested in Section 4 and described in greater detail in Appendix F.2. We train and validate these architectures with the WIDER FACE dataset [47]. Versions Quant, A, B, and C of the RNNPool-Face use RNNPoolLayer of hidden dimensions 4, 4, 6 and 16, respectively.

Table 4 compares validation Mean Average Precision (MAP) for easy, medium, and hard subsets. MAP is a standard metric for face detection and measures the mean area under the precision-recall curve. We report MAP scores for baselines based on the official open-source code or pre-trained models. For Eagle-Eye [52], we re-implemented the method as the source code was not available. For EXTD [48], we report MAdds of the EXTD-32 version - the computationally cheapest. EXTD and LFFD [18] are accurate but are computationally expensive. In contrast, RNNPool-Face-C achieves better MAP in the easy and medium subsets despite using $\sim 4.5\times$ less compute and $\sim 3\times$ less RAM.

FaceBoxes [49] and Eagle-Eye reduce MAdds and peak memory usage by aggressively down-sampling the image or by decreasing the number of channels leading to inaccurate models. In contrast, RNNPool-Face-A and RNNPool-Face-B achieve significantly higher MAPs than these methods while still ensuring smaller MAdds and peak RAM usage. We also compare MAP scores for images that have ≤ 3 faces, which is a more realistic face-detection setting for tiny devices. Here also, RNNPool-Face-C is more accurate than all the baselines. Finally, RNNPool-Face-Quant uses byte quantization to reduce the model size so it can be deployed on Cortex-M4 devices which typically have $\leq$ 256 KB RAM, while still having $>$ 0.80 MAP accuracy on images with ≤ 3 faces. See Appendix I for a qualitative evaluation of our method against the baselines.

5.4 RNNPool based Model for ARM Cortex-M4 Microcontrollers

Finally, we develop a face detection model for conference/class room settings that can be deployed on ARM Cortex-M4 class devices. To this end, we develop a more compact version of the face detection model, RNNPool-Face-M4 (Table 15 in Appendix F.2), which has only 4 MBConv blocks. For

Table 5: Comparison of resources and MAP on the SCUT-HEAD dataset. RNNPool-Face-M4 can be effectively deployed on an M4 device with <256 KB RAM in contrast to MobileNetV2-SSDLite low-cost detection model.

Model	MAP	Peak RAM	MAdds	Model Size
MobileNetV2-SSDLite	**0.63**	3.51 MB	540M	11.32 MB
RNNPool-Face-M4	0.58	**188 KB**	**70M**	**160 KB**

further reduction in MAdds and model-size, we train the RNNPool parameters to be sparse. That is, $\mathbf{W}$ matrix of RNN_1 is 50% non-zeros while the rest of the matrices in RNNPool are 30% non-zeros.

To not overshoot RAM for storing input image, we use $320\times240\times1$ monochrome images for training and testing. For evaluation, we first train on the WIDER FACE dataset and then fine-tune on the SCUT-HEAD dataset [35] which consists of images in conference/class rooms. We then use the SeeDot [12] compiler to quantize our model to 8 bits and generate C code for deployment. Table 5 compares the resource requirements and MAP on the SCUT-HEAD validation set (random 80%-20% split) of RNNPool-Face-M4 against a similarly trained MobileNetV2-SSDLite model which is a state-of-the-art architecture for low-cost detection.

Note that MobileNetV2-SSDLite cannot be deployed on a Cortex-M4 device even with 8-bit quantization as the peak RAM requirement is much more than the 256 KB limit of the device. RNNPool-Face-M4 model processes a single image in 10.45 seconds on an ARM Cortex-M4 microcontroller based STM32F439-M4 device clocked at 168 MHz.

6 Conclusions

In this paper, we proposed RNNPool, an efficient RNN-based pooling operator that can be used to rapidly downsample activation map sizes thus significantly reduce inference-time memory and compute requirements for a variety of standard CNNs. Due to syntax level similarity with pooling layers, we can use RNNPool in most existing CNN based architectures. These replacements retain accuracy for tasks like image classification and visual wake words. Our S3FD based RNNPool model for face detection provided accurate models that can be deployed on tiny Cortex-M4 microcontrollers. Finally, we showed with Proposition 1 that calculations of minimum memory requirement for standard architectures can be made rigorous and demonstrate that despite such optimizations of standard CNNs, RNNPool based models can be significantly more efficient in terms of inference-time working memory. Using neural architecture search for RNNPool based models to further reduce inference cost is an immediate and interesting direction.

Broader Impact

Pros: ML models are compute-intensive and are typically served on power-intensive cloud hardware with a large resource footprint that adds to the global energy footprint. Our models can help reduce this footprint by (a) allowing low power edge sensors with small memory to analyze images and admit only interesting images for cloud inference, and (b) reducing the inference complexity of the cloud models themselves. Further, edge-first inference enabled by our work can reduce reliance on networks and also help provide privacy guarantees to end-user. Furthermore, vision models on tiny edge devices enables accessible technologies, e.g., Seeing AI [33] for people with visual impairment.

Cons: While our intentions are to enable socially valuable use cases, this technology can enable cheap, low-latency and low-power tracking systems that could enable intrusive surveillance by malicious actors. Similarly, abuse of technology in certain wearables is also possible.

Again, we emphasize that it depends on the user to see the adaptation to either of these scenarios.

Acknowledgements

We are grateful to Shikhar Jaiswal and Aayan Kumar for their assistance in the deployment of RNNPool models on Cortex-M4 devices. We also thank Sahil Bhatia, Ali Farhadi, Sachin Goyal, Max Horton, Sham Kakade and Ajay Manchepalli for helpful discussions and feedback. Aditya Kusupati did a part of this work during his research fellowship at Microsoft Research India.

References

[1] D. Acuna, H. Ling, A. Kar, and S. Fidler. Efficient interactive annotation of segmentation datasets with polygon-rnn++. In *The IEEE conference on Computer Vision and Pattern Recognition*, pages 859–868, 2018.

[2] S. Bell, C. Lawrence Zitnick, K. Bala, and R. Girshick. Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition*, June 2016.

[3] H. Cai, L. Zhu, and S. Han. ProxylessNAS: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

[4] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[5] M. Cho and D. Brand. Mec: memory-efficient convolution for deep neural network. In *International Conference on Machine Learning*, pages 815–824. JMLR. org, 2017.

[6] A. Chowdhery, P. Warden, J. Shlens, A. Howard, and R. Rhodes. Visual wake words dataset. *arXiv preprint arXiv:1906.05721*, 2019.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *The IEEE conference on Computer Vision and Pattern Recognition*, pages 248–255. Ieee, 2009.

[8] D. K. Dennis, Y. Gaurkar, S. Gopinath, S. Goyal, C. Gupta, M. Jain, S. Jaiswal, A. Kumar, A. Kusupati, C. Lovett, S. G. Patil, O. Saha, and H. V. Simhadri. EdgeML: Machine Learning for resource-constrained edge devices. URL https://github.com/Microsoft/EdgeML.

[9] T. Gale, E. Elsen, and S. Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

[10] Y. Gong, L. Wang, R. Guo, and S. Lazebnik. Multi-scale orderless pooling of deep convolutional activation features. In *European Conference on Computer Vision*, pages 392–407. Springer, 2014.

[11] Google. ML for mobile and edge devices - tensorflow lite. URL https://www.tensorflow.org/lite.

[12] S. Gopinath, N. Ghanathe, V. Seshadri, and R. Sharma. Compiling kb-sized machine learning models to tiny iot devices. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 79–95, 2019.

[13] A. Gural and B. Murmann. Memory-optimal direct convolutions for maximizing classification accuracy in embedded applications. In *International Conference on Machine Learning*, pages 2515–2524, 2019.

[14] S. Han, J. Lin, K. Wang, T. Wang, and Z. Wu. Solution to Visual Wakeup Words Challenge'19 (first place). URL https://github.com/mit-han-lab/VWW.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *The IEEE transactions on Pattern Analysis and Machine Intelligence*, 37 (9):1904–1916, 2015.

[16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *The IEEE conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[17] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *The IEEE International Conference on Computer Vision*, pages 1389–1397, 2017.

[18] Y. He, D. Xu, L. Wu, M. Jian, S. Xiang, and C. Pan. LFFD: A light and fast face detector for edge devices. *arXiv preprint arXiv:1904.10633*, 2019.

[19] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[20] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[21] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *The IEEE conference on Computer Vision and Pattern Recognition*, pages 4700–4708, 2017.

[22] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[23] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *International Symposium on Computer Architecture*, pages 1–12, 2017.

[24] D. Kim, J.-Y. Choi, and J.-E. Hong. Evaluating energy efficiency of internet of things software architecture based on reusable software components. *International Journal of Distributed Sensor Networks*, 13(1):1550147716682738, 2017.

[25] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[26] A. Kusupati, M. Singh, K. Bhatia, A. Kumar, P. Jain, and M. Varma. FastGRNN: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network. In *Advances in Neural Information Processing Systems*, pages 9017–9028, 2018.

[27] A. Kusupati, V. Ramanujan, R. Somani, M. Wortsman, P. Jain, S. Kakade, and A. Farhadi. Soft threshold weight reparameterization for learnable sparsity. In *International Conference on Machine Learning*, 2020.

[28] L. Lai, N. Suda, and V. Chandra. Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus. *arXiv preprint arXiv:1801.06601*, 2018.

[29] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[30] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision*, pages 740–755. Springer, 2014.

[31] N. Liu, J. Han, and M.-H. Yang. Picanet: Learning pixel-wise contextual attention for saliency detection. In *The IEEE Conference on Computer Vision and Pattern Recognition*, June 2018.

[32] A. Mead. Review of the development of multidimensional scaling methods. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 41(1):27–39, 1992.

[33] Microsoft. Seeing AI. URL https://www.microsoft.com/en-us/ai/seeing-ai.

[34] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[35] D. Peng, Z. Sun, Z. Chen, Z. Cai, L. Xie, and L. Jin. Detecting heads using feature refine net and cascaded multi-scale architecture. *arXiv preprint arXiv:1803.09256*, 2018.

[36] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[37] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *The IEEE conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[38] A. Shaw, D. Hunter, F. Iandola, and S. Sidhu. SqueezeNAS: Fast neural architecture search for faster semantic segmentation. In *ICCV Neural Architects Workshop*, 2019.

[39] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning*, pages 1139–1147, 2013.

[40] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *The IEEE conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[41] M. Tan and Q. Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114, 2019.

[42] F. Visin, K. Kastner, K. Cho, M. Matteucci, A. Courville, and Y. Bengio. Renet: A recurrent neural network based alternative to convolutional networks. *arXiv preprint arXiv:1505.00393*, 2015.

[43] J. Wang, Y. Yang, J. Mao, Z. Huang, C. Huang, and W. Xu. CNN-RNN: A unified framework for multi-label image classification. In *The IEEE conference on Computer Vision and Pattern Recognition*, pages 2285–2294, 2016.

[44] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han. Haq: Hardware-aware automated quantization with mixed precision. In *The IEEE conference on Computer Vision and Pattern Recognition*, pages 8612–8620, 2019.

[45] W. Xie, A. Noble, and A. Zisserman. Layer recurrent neural networks. 2016.

[46] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems*, pages 802–810, 2015.

[47] S. Yang, P. Luo, C.-C. Loy, and X. Tang. Wider face: A face detection benchmark. In *The IEEE conference on Computer Vision and Pattern Recognition*, pages 5525–5533, 2016.

[48] Y. Yoo, D. Han, and S. Yun. EXTD: Extremely tiny face detector via iterative filter reuse. *arXiv preprint arXiv:1906.06579*, 2019.

[49] S. Zhang, X. Zhu, Z. Lei, H. Shi, X. Wang, and S. Z. Li. Faceboxes: A CPU real-time face detector with high accuracy. In *The IEEE International Joint Conference on Biometrics*, pages 1–9. IEEE, 2017.

[50] S. Zhang, X. Zhu, Z. Lei, H. Shi, X. Wang, and S. Z. Li. S3fd: Single shot scale-invariant face detector. In *The IEEE International Conference on Computer Vision*, pages 192–201, 2017.

[51] Q. Zhao, S. Lyu, B. Zhang, and W. Feng. Multiactivation pooling method in convolutional neural networks for image recognition. *Wireless Communications and Mobile Computing*, 2018, 2018.

[52] X. Zhao, X. Liang, C. Zhao, M. Tang, and J. Wang. Real-time multi-scale face detector on embedded devices. *Sensors*, 19(9):2158, 2019.