# RNNPool: Efficient Non-linear Pooling for RAM Constrained Inference

Oindrila Saha[1] Aditya Kusupati[2] Harsha Vardhan Simhadri[1] Manik Varma[1] Prateek Jain[1]

[1]Microsoft Research, India
[2]University of Washington, USA

## Abstract

*Pooling operators are key components in most Convolutional Neural Networks (CNNs) as they serve to downsample images, aggregate feature information, and increase receptive field. However, standard pooling operators reduce the feature size gradually to avoid significant loss in information via gross aggregation. Consequently, typical CNN architectures are designed to be deep, computationally expensive and challenging to deploy on RAM constrained devices. We introduce* RNNPool*, a novel pooling operator based on Recurrent Neural Networks (RNNs), that efficiently aggregate features over large patches of an image and rapidly downsamples its size. Our empirical evaluation indicates that an* RNNPool *layer(s) can effectively replace multiple blocks in a variety of architectures such as MobileNets [7], DenseNet [5] and can be used for several vision tasks like image classification and face detection. That is,* RNNPool *can significantly decrease computational complexity and peak RAM usage for inference while retaining comparable accuracy. Further, we use* RNNPool *to construct a novel real-time face detection method that achieves state-of-the-art MAP within computational budget afforded by a tiny Cortex M4 microcontroller with* $\sim 256$ *KB RAM.*

## 1. What is RNNPool?

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

### 1.1. 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 – $\text{RNN}_1$ and $\text{RNN}_2$ of hidden dimension $h_1$ and $h_2$ respectively – to sweep the patch horizontally and vertically to produce a summary of size $1 \times 1 \times 4h_2$. While it is possible to use GRU [1] or LSTM [4] for the RNNs in RNNPool, we use FastGRNN [6] for its compact size and fewer FLOPs.

An RNNPoolLayer consists of a single RNNPool operator strided over an input activation map. Note that there are only two RNNs ($\text{RNN}_1$ & $\text{RNN}_2$) in an RNNPool operator, thus weights are shared for both the row-wise and column-wise passes ($\text{RNN}_1$) and all bi-directional passes ($\text{RNN}_2$) across every instance of RNNPool in an RNNPoolLayer. Further, RNNPoolLayer also takes two more parameters into account: patch size and the stride.

### 1.2. Comparing with Pooling Operators

We contrast the down-sampling power of RNNPool against standard pooling operators. That is, we investigate if the pooling operators maintain accuracy for a downstream task even when the pooling receptive field is large. To this end, we consider the image classification task with CIFAR-10 dataset but the pooling operator is required to down-sample the input $32 \times 32$ image to a $1 \times 1$ voxel in *one go* i.e. both patch size and stride are 32. This is followed by a fully connected (FC) layer. RNNPool achieves an accuracy of **70.63%**, while convolution layer, max pooling and average pooling's accuracy are $53.13\%$, $20.04\%$ and $26.53\%$, respectively. This demonstrates the modeling power of the RNNPool operator over other pooling methods. This shows that if we pool aggressively (using standard operators), the accuracy drop is large. So standard pooling operators do not suffice for our purpose of RAM reduction. We show how RNNPool, when combined with standard models, leads to comparable accuracy on multiple

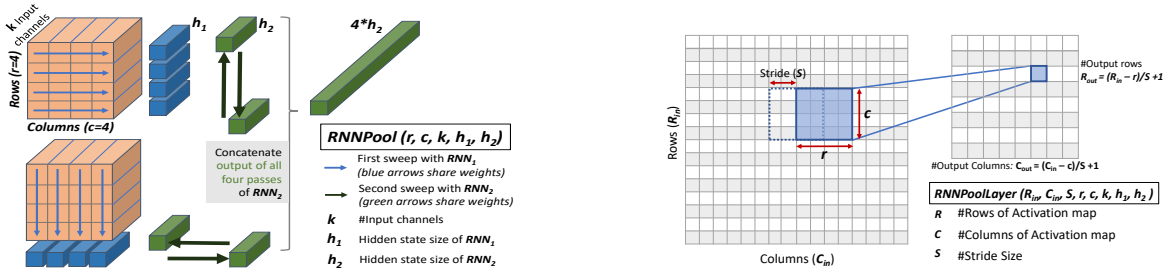


Figure 1: (left) An RNNPool operator. (right) An RNNPoolLayer composed with strided RNNPool operators with shared weights.

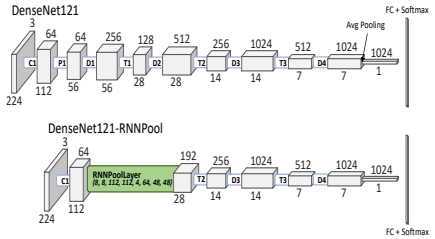


Figure 2: **DenseNet121-**RNNPool: obtained by replacing P1, D1, T1 and D2 blocks in DenseNet121 with an RNNPoolLayer.

tasks but with much smaller RAM/FLOPs requirements in Tables 1, 2 and 3.

## 2. How to use the RNNPoolLayer?

RNNPool can be used to modify several state-of-the-art architectures to reduce their working memory as well as computational requirements. Typically, such modifications involve replacing one or more stacks of convolutional and pooling layers of the "base" (original) architecture with an RNNPoolLayer and retraining from scratch.

### 2.1. Replacement for a Sequence of Blocks

Consider the DenseNet121 [5] 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 maximum working memory and FLOPs as it deals with large activation maps that are yet to be downsampled. Furthermore, the presence of residual connections between all 6 layers within each dense block exacerbates the memory management problem. We can use an RNNPoolLayer to rapidly downsample the image size and bypass intermediate large activations. In DenseNet121, we

Table 1: Resources vs accuracy for ImageNet-1K.

| Method | Peak RAM | #Params | FLOPs | Accuracy (%) |
|---|---|---|---|---|
| MobileNetV1 | 3.06MB | 4.2M | 569M | **69.52** |
| MobileNetV1-RNNPool | **0.77MB** | **4.1M** | **417M** | 69.39 |
| MobileNetV2 | 2.29MB | 3.4M | 300M | **71.81** |
| MobileNetV2-RNNPool | **0.24MB** | **3.2M** | **226M** | 70.14 |
| EfficientNet-B0 | 2.29 MB | 5.3M | 390M | **76.30** |
| EfficientNet-B0-RNNPool | **0.25 MB** | **5.2M** | **330M** | 72.47 |

Table 2: Comparison of memory requirement, # parameters and validation MAP obtained by different methods for Face Detection on the WIDER FACE dataset [8]. RNNPool-Face-C is able to achieve higher accuracy than the baselines despite using 3× less RAM and 4.5× less FLOPs. RNNPool-Face-Quant enables deployment on Cortex M4 class devices with 6-7% accuracy gains over the cheapest baselines.

| Method | Peak RAM | Parameters | FLOPs | MAP | | | MAP for ≤ 3 faces | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | Easy | Medium | Hard | Easy | Medium | Hard |
| EagleEye[12] | 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** |
| FaceBoxes[10] | 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 |
| EXTD[9] | 18.75 MB | **0.07M** | 8.49G | 0.90 | 0.88 | **0.82** | 0.93 | 0.93 | 0.91 |
| LFFD[3] | 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** |
| RNNPool-Face-Quant | 225 KB | 0.07M | 0.12G | 0.80 | 0.78 | 0.53 | 0.84 | 0.83 | 0.81 |

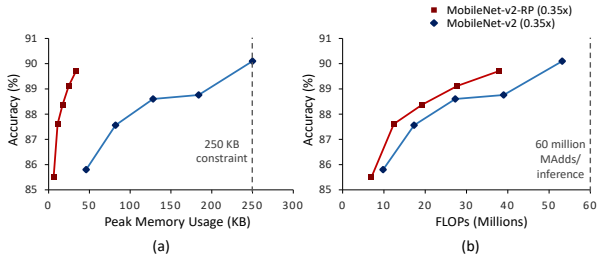


Figure 3: Visual Wake Word [2]: MobileNetV2-RNNPool requires 8× less RAM and 40% less compute than baselines. We also cap our number of parameters to be ≤ 250K instead of 290K of MobileNetV2 (0.35×).

can replace 4 blocks - P1, D1, T1, D2 - spanning *39* layers with a single RNNPoolLayer to reduce the activation map from size 112 × 112 to 28 × 28 (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 contribute greatly to the reduction in working memory size as well as the number of computations. Table 1 and Figure 3, illustrate the reduction of resources wrt baselines on classification tasks while using RNNPool following above strategy, while not compromising on accuracy.

### 2.2. New Architectures for Face Detection

Using RNNPoolLayer, we design new architectures for face detection that achieve higher MAP scores than state-of-the-art and are compact enough for real-time face detection on weak microcontrollers. We start with the structure of S3FD [11]. Instead of applying one convolution layer and

then applying RNNPoolLayer, we directly down-sample the image by a factor of $1/4$ via RNNPoolLayer. This is critical as for the smallest faces, the anchor box size is set as $16 \times 16$ and the required stride is 4. Further, for efficiency, we use depthwise separable convolutions followed by point-wise convolutions before using inverted residual (MBConv) blocks. We also create an architecture which can fit within the resource constraints to be deployed on a Cortex M4 microcontroller i.e. it has the peak RAM usage as $\leq$ 256 KB and inference cost $\leq$ 128 MFLOPs (latency $\leq$ 1s) while having a competitive MAP (Table 2).

## References


[1] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[2] Aakanksha Chowdhery, Pete Warden, Jonathon Shlens, Andrew Howard, and Rocky Rhodes. Visual wake words dataset. *arXiv preprint arXiv:1906.05721*, 2019.

[3] Yonghao He, Dezhong Xu, Lifang Wu, Meng Jian, Shiming Xiang, and Chunhong Pan. LFFD: A light and fast face detector for edge devices. *arXiv preprint arXiv:1904.10633*, 2019.

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

[5] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[6] Aditya Kusupati, Manish Singh, Kush Bhatia, Ashish Kumar, Prateek Jain, and Manik 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.

[7] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[8] Shuo Yang, Ping Luo, Chen-Change Loy, and Xiaoou Tang. Wider face: A face detection benchmark. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5525–5533, 2016.

[9] YoungJoon Yoo, Dongyoon Han, and Sangdoo Yun. EXTD: Extremely tiny face detector via iterative filter reuse. *arXiv preprint arXiv:1906.06579*, 2019.

[10] Shifeng Zhang, Xiangyu Zhu, Zhen Lei, Hailin Shi, Xiaobo Wang, and Stan Z Li. Faceboxes: A CPU real-time face detector with high accuracy. In *2017 IEEE International Joint Conference on Biometrics (IJCB)*, pages 1–9. IEEE, 2017.

[11] Shifeng Zhang, Xiangyu Zhu, Zhen Lei, Hailin Shi, Xiaobo Wang, and Stan Z Li. S3fd: Single shot scale-invariant face detector. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 192–201, 2017.


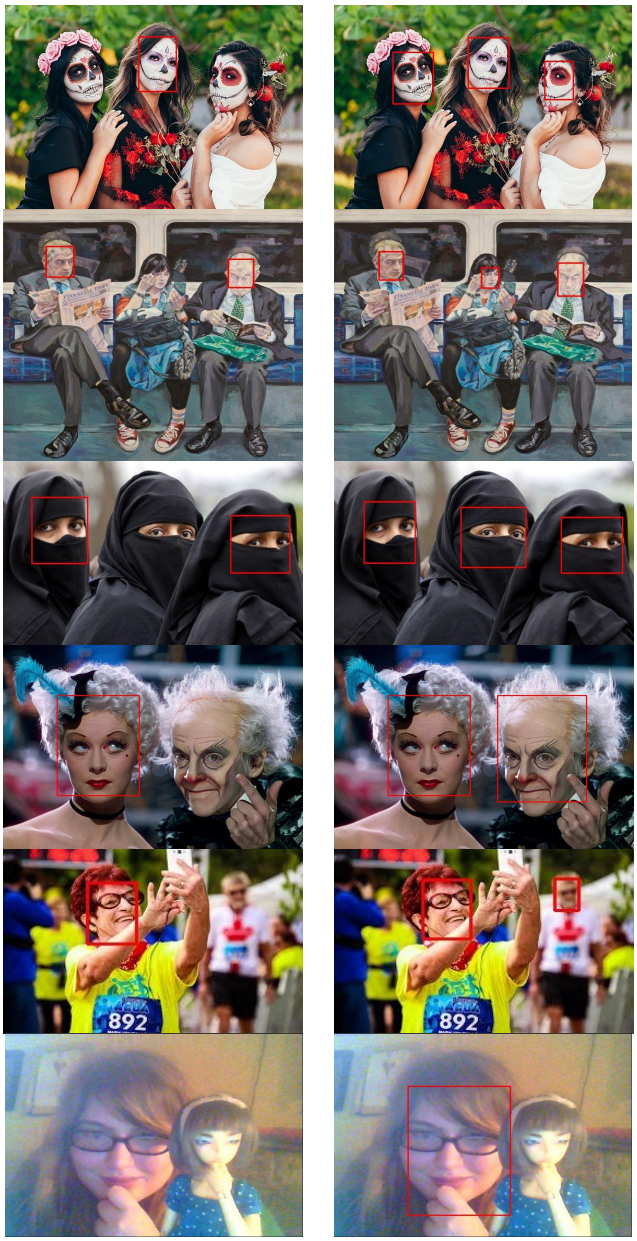


Figure 4: Comparison of performance on test images with Eagle-Eye and RNNPool-Face-Quant. The confidence threshold is set at 0.6 for both models. EagleEye misses faces when there is makeup, occlusion, blurriness and in grainy pictures, while our method is detecting them. However, in the case of some hard faces, RNNPool-Face-Quant still misses a few of them or doesn't give out a bounding box containing the full face.


[12] Xu Zhao, Xiaoqing Liang, Chaoyang Zhao, Ming Tang, and Jinqiao Wang. Real-time multi-scale face detector on embedded devices. *Sensors*, 19(9):2158, 2019.