# MemNet: Memory-Efficiency Guided Neural Architecture Search with Augment-Trim learning

Peiye Liu[1], Bo Wu[2], Huadong Ma[1], Pavan Kumar Chundi[2], Mingoo Seok[2],
[1] Beijing University of Posts and Telecommunications,Beijing, China
[2] Columbia University, New York, US
{liupeiye, mhd}@bupt.edu.cn, {bo.wu,pc2769,ms4415}@columbia.edu

## Abstract

*Recent studies on automatic neural architectures search have demonstrated significant performance, competitive to or even better than hand-crafted neural architectures. However, most of the existing network architecture tend to use residual, parallel structures and concatenation block between shallow and deep features to construct a large network. This requires large amounts of memory for storing both weights and feature maps. This is challenging for mobile and embedded devices since they may not have enough memory to perform inference with the designed large network model. To close this gap, we propose* ***MemNet****, an augment-trim learning-based neural network search framework that optimizes not only performance but also memory requirement. Specifically, it employs memory consumption based ranking score which forces an upper bound on memory consumption for navigating the search process. Experiment results show that, as compared to the state-of-the-art efficient designing methods, MemNet can find an architecture which can achieve competitive accuracy and save an average of* $24.17\%$ *on the total memory needed.*

## 1. Introduction

Deep Neural Networks (DNNs) have demonstrated state-of-the-art results in image classification [31, 15, 7] and object detection [4, 19, 27, 18]. However, those state-of-the-art neural network architectures are extremely deep and also highly complicated, making it a non-trivial task to design them manually. This has drawn researchers' attention to Neural Architecture Search (NAS), which involves techniques to construct neural networks without the need for profound domain knowledge [22, 26, 1, 36, 37].

In order to make neural networks efficient, we must consider the target platform that performs inference. Nowadays, we have mainly two platforms, namely cloud (data center) and mobile computing. A neural network which is to be run on a cloud or desktop computer can leverage massive computing resources. Therefore, most of the cloud and desktop-based works focus on optimizing the speed of the search process and the accuracy of neural networks [22, 26, 1, 36, 37, 30, 5, 25, 30, 14, 29].

The mobile computing platform on the other hand needs a network that takes less memory and power. Towards that, some works employ Pareto-Optimal algorithm [8] to optimize a hardware-oriented criterion (e.g., latency and energy consumption) [10, 3, 21]. Considering that on-chip memory access requires 3 or 4 orders of magnitudes less energy and delay compared to off-chip FLASH and DRAM [33], optimizing a network to fit entirely in the tiny on-chip memory ($< 5$MB) is the ultimate target.

To achieve that memory efficient architecture, our work intends to optimize runtime memory consumption in NAS. During inference, memory consumption of a network includes two major parts: network parameters and intermediate representation. Network parameters represent the weights and the intermediate representation corresponds to feature maps. Different from network parameters, the storage of intermediate representation can be released immediately after being used. However, it's hard to determine a specific time for releasing the used intermediate representations in parallel and skip structure based network architectures. We would therefore require an NAS model that pays more attention to memory efficient network structures while increasing the size of the network.

We address the challenge of NAS for memory efficient neural network (MemNet) through the following set of contributions.

- Estimating the dynamic intermediate memory consumption using data lifetime
- Augment-trim learning to pick efficient candidate architectures and trim redundant connection in each search iteration
- Memory optimization based loss function to train a rel-

ative ranking controller for extracting the correlation between candidate architectures and choosing top $K$ memory efficient candidates

Thorough experiments demonstrate the capability of MemNet on choosing better memory efficient architectures and achieving an outstanding performance with less memory consumption.

The reminder of this paper is organized as follows. Section 2 reviews the previous efficient neural architecture design methods. Section 3 provides details on the proposed MemNet. Section 4 provides the experimental results. Finally, in Section 5, we will conclude the paper.

## 2. Related Work

### 2.1. Human-crafted Design

The need for performing inference operation with high-quality DNNs models on a resource-constrained mobile device has been steadily increasing. This has motivated a number of studies on reducing the size and computational complexity of DNNs without compromising accuracy performance. A thread of works have explored the use of filters with a smaller kernel size and concatenated several of them to emulate a large filter [6, 16, 2, 35, 23, 28]. For example, GoogLeNet adopts one $1 \times N$ and one $N \times 1$ convolutions to replace the $N \times N$ convolution [32]. Similarly, it is also proposed to decompose a 3-D convolution to a set of 2-D convolutions. For example, MobileNet decomposes the original $N \times N \times M$ convolution ($N$ is the kernel size and $M$ is the filter number) to one $N \times N \times 1$ convolution and one $1 \times 1 \times M$ convolution [9]. This can reduce the filter-related computations from $N \times N \times M \times I \times O$ ($I$ is the input channels and $O$ is the convolution output size) to $N \times N \times M \times O + M \times I \times O$. In addition, SqueezeNet adopts a fire module that first squeezes the network with $1 \times 1$ convolution filters and then expands it with multiple $1 \times 1$ and $3 \times 3$ convolution filters [12]. ShuffleNet utilizes the point-wise group convolution to replace the $1 \times 1$ filter for further reducing computation complexity [34].

### 2.2. Neural Architecture Search

Neural architecture search (NAS) has recently emerged to automatically create high-performance networks. Zoph *et* al. presented a seminal work where they introduce the Reinforcement Learning (RL) for NAS [36]. Since then several works proposed NAS models. Recently, Dong *et* al. proposed the PPP-Net framework [3]. The framework considers both inference time and accuracy. It treats the selection of neural network candidates as a multi-objective optimization [8] problem and chooses the neural architecture in the Pareto front area. However, the framework adopts CondenseNet [11] which has a large amount of runtime memory consumption. Another disadvantage is that the algorithm requires manual intervention to pick from the selected Pareto front area in each search iteration. Hsu *et* al. [10] proposed MONAS framework employing a reward function of prediction accuracy and power consumption. It successfully constructs a low power neural architecture. However, it considers only a small search space consisting of existing networks, namely AlexNet [15] and CondenseNet [11], and their variants. Michel *et* al. proposed the DVOLVER framework [21]. It only considers network parameter minimization as an additional objective and could produce a network with large memory consumption.

## 3. Proposed Method

Our goal is to find a neural architecture that achieves the best trade-off between inference accuracy and upper bound of runtime memory consumption. Figure 1 depicts the overview of the proposed framework, MemNet. The typical process of the framework is as follows. It first generates many neural network candidates, each either augmented or trimmed from the neural network found/selected from the previous iteration. Then, it asks the controller to predict the top-$K$ candidates in term of the customized neural network merit score that considers both accuracy performance and memory requirement. Finally, we train the top-$K$ candidates, compare their accuracy performances and memory requirements, and select the best neural architecture, based on the customized merit score, for the current search iteration. In addition, the results of the training are used to train the controller for the use of subsequent iterations.

In the following subsections we elaborate on the proposed method while discussing the differences with prior work.

### 3.1. Neural Network Candidate Generation via Augmentation and Trim

In NAS, each search iteration starts with generating multiple neural network candidates from the neural network found/selected in the previous iteration. In most of the existing NAS frameworks, the new candidates are almost always more complex, e.g., having more layers or each layer becoming wider. However, as we consider both accuracy and memory usage, we generate candidates not only by augmenting but also trimming the neural network from previous iteration. This allows NAS to explore a wider design space including both accuracy and memory optimal candidates. Inspired by Evolutionary Algorithm [26], we employ a multiple-objective hierarchical neural architecture search framework, in which we construct different cell structures, and then connect them sequentially, in order to generate the final DNNs. A cell is constructed of B blocks arranged in a Directed Acyclic Graph (DAG). Each block contains two individual layers and one connection operation, map-

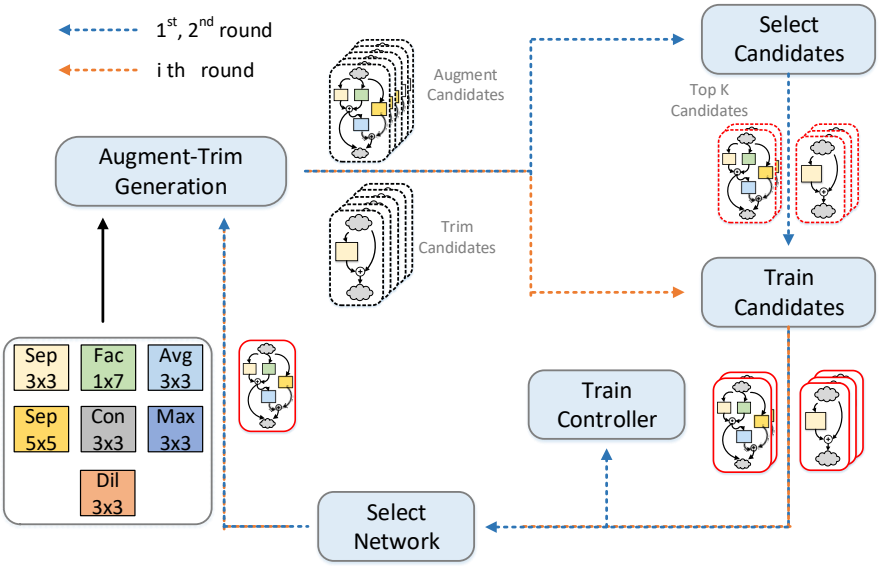


Figure 1: **The framework of MemNet**. It consists of major operations and intermediate candidates. It starts from the previously selected network and ends on generating a new memory efficient architecture. The solid rectangle denotes the trained network and the dotted rectangle denotes untrained architecture candidates. The red rectangle denotes the selected candidates.

ping from two input tensors to one output tensor, which can be represented by a 5-tuple, $(IP_1, IP_2, O_1, O_2, C)$. Specifically, $IP_1$ and $IP_2$ are the input location of two layers, which includes the output of the previous blocks in this cell and the output from previous cell. $C$ shows the connection method for the output of layers, including addition and concatenation. At each layer, $O_1$ and $O_2$ represent one of seven possible operations is chosen. In order to construct the best architecture satisfying the trade-offs between the performance and memory usage, we adopt seven layer operation types from previous efficient architecture design [17].

- 3 x 3 convolution
- 3 x 3 depthwise convolution
- 5 x 5 depthwise convolution
- 1 x 7 followed by 7 x 1 convolution
- 3 x 3 average pooling
- 3 x 3 max pooling
- 3 x 3 dilated convolution

In each search iteration, we apply 'augment' generation operation with the above 5-tuple block to increase a new block to the original cell structure. Therefore, an individual cell $C$ with $B$ blocks can be represented by $[(IP_1, IP_2, O_1, O_2, C)_1, ..., (IP_1, IP_2, O_1, O_2, C)_B]$. Then we stack the normal cells which don't change the feature map size during inference, and reduction cells, which shrink the feature map size to half of the input size, to formulate the final network architecture. Therefore, let the 'augment' space of possible structures for $b$ blocks be $S_r$:

$$|S_r| = |p|^2 * |o|^2 * |c|^2 \tag{1}$$

where $p$ denotes the number of available candidate input locations, $o$ represents the seven candidate operations and $c$ denotes the connection methods. Considering that the cell structure sharing strategy will bring possible redundant connections and blocks, we also adapt to generate a 'trim' search space to construct a more efficient network architecture. In each iteration, we construct the 'trim' search space by two operations. Firstly we transform each individual layer in one block of one cell to identity operation for a trim architecture search space, shown in Figure 2. And considering the explosive memory growth caused by the uniformity concatenation at the end of a cell, we also cut a single connection between the block to the end for another 'trim' architecture search space. Therefore, let the 'trim' space of possible structures for $b$ blocks be $S_t$:

$$|S_t| = |b - 1| * N_{cell} * 2 + N_{con} \tag{2}$$

where $N_{cell}$ denotes the number of cells in the whole network and $N_{con}$ denotes the number of existing connection between block and end of the cell. It's obvious that the magnificent difference variation tendency between the 'augment' and 'trim' search space brings more difficult to the

candidates selection process. Therefore, we arrange a ranking score based memory-guided loss function and network-cross ranking controller to precisely estimate a ranking performance among the total $S_t + S_r$ candidates.

### 3.2. Relative Ranking Controller

The existing NAS works employ controllers that take one neural network candidate as input and produce its merit score as output. Each candidate is then evaluated and the candidate achieving the highest score is selected. In our MemNet, however, such simple controller architecture tends to be less effective. This is because MemNet generates the candidates both by augmentation and trimming and thus the resulted candidates requires the controller to evaluate significantly more diverse neural network architectures. To address this issue, we adopt an RNN based controller that takes multiple neural networks as input and produce the rankings among them as output.

Considering that there is totally different topology and variation tendency in the search spaces, it's hard to conclude correct prediction only based on single network architecture. Therefore, we decide to extract the correlation between the different candidate architectures to estimate a ranking result. To accurately locate the best architectures satisfying the trade-offs between the network performance and memory consumption in the prediction process, the controller is able to learn efficiently from a few data points and deal with variable-size inputs (the size of the cell structure will increase in successive iterations). Therefore, we construct a Network-Cross Ranking network to deal with the prediction task in our MemNet, as shown in Figure 4. The ranking network consists of two individual Recurrent Neural Networks (RNN). The first encoder RNN is responsible for dealing with the variable-length input block structure features and encodes the various block features to an entire network feature. Then the second network-cross ranking RNN will receive K-output of the last hidden state of the previous RNN and ranks the score of the multiple network architecture according to the multi-objective loss function mentioned in Section 3.2.

In iteration $i$, each candidate architecture $N_i \in \{n_1, n_2, ..., n_s\}$ will be embedded to a set of block features $f_b \in \{f_1, f_2, .., f_c\}$, where $c$ denotes the number of blocks in each cell. Then, the block features $f_b$ will be feed to encoder-RNN from top of the network to bottom to generate an entire network feature $f_n$ with the same length with the size of encoder-RNN's hidden layer. After obtaining all of the $f_n$, they will be sent to ranking-RNN to dig the correlation between those candidate architectures and give the estimated ranking score. After training the top $K$ candidate architectures, all of the previously trained architectures will be utilized to retrain a new network-cross ranking controller for the next iteration.

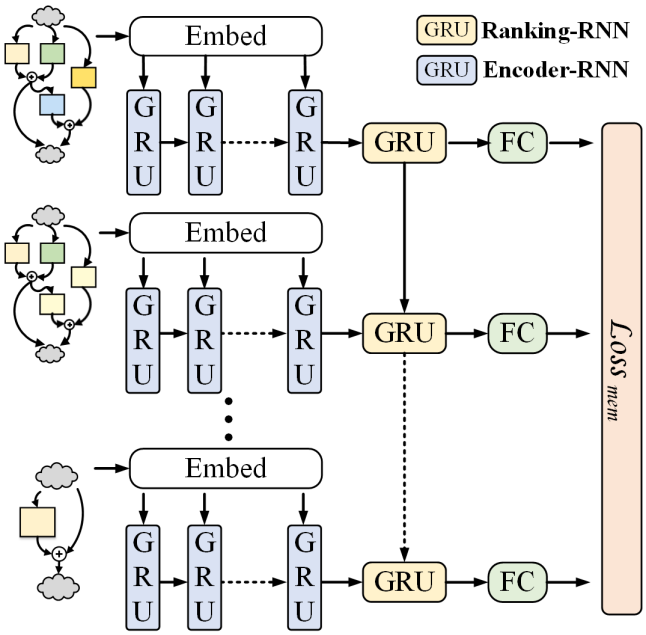


Figure 2: The training architecture of the Relative Ranking controller, which includes an encoder-RNN and a ranking-RNN.

We also incorporate the expected network memory consumption into the normal loss function by multiplying a scaling factor $\lambda_s$ which controls the trade-offs between performance and memory consumption. In general, we set the $\lambda_s = \frac{\lambda_\beta}{\lambda_\alpha}$ for training the network-cross ranking controller. Therefore, the final multi-objective loss function is defined as:

$$Loss_{mem} = L_{CE} + \frac{\lambda_\beta}{\lambda_\alpha}(L_{MSE}(P) + L_{MSE}(M)) \quad (3)$$

where $L_{CE}$ denotes the Cross-entropy (CE) loss and $L_{MSE}$ denotes the Mean Square Error (MSE) loss function.

### 3.3. Accuracy and Memory Optimization

We aim to search for a memory efficient architecture, through a criterion which balances the network performance and running memory consumption. We propose a custom score that is based on both accuracy performance and memory efficiency. In each search iteration of NAS, we select a neural network, among multiple candidates, based on the merit of neural networks. In the existing NAS frameworks, the merit (score) is often defined solely by the accuracy performance.

Augment-trim strategy brings more variation tendency on both network performance and memory consumption in the search process which is unlike single growth generation. In 'augment' search space, the performance usually increases with increase in memory consumption. On

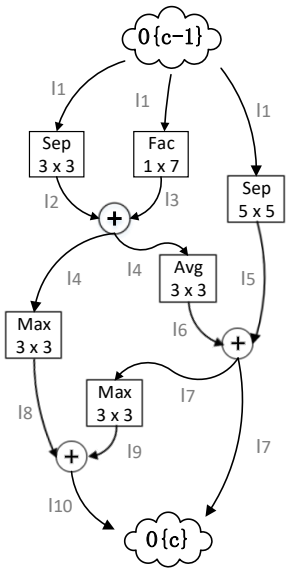


Figure 3: A sample of neural architecture search process on lifetime based intermediate representation calculation. The Sep denotes the depthwise convolution, the Fac denotes 1 $\times$ 7 followed 7 $\times$ 1 convolution, the Avg denotes average pooling, and Max denotes max pooling layer.

the other hand, in the trim search space, there is no clear positive correlation between performance and memory consumption. Therefore, we combine the normalized variation value of performance and memory consumption to build a customized weighted ranking score.

Given a neural architecture $n$, let $a$ denote its accuracy on the target task, $m$ denote the maximum intermediate representation memory consumption calculated above, $p$ denote the inference network parameter consumption. The following expression gives the customized weighted ranking score which is used to automatically select the best architecture meeting the trade-offs between the performance and memory consumption in the selection.

$$\lambda_\alpha \frac{a_{cur} - a_{pre}}{a_{pre}} + \lambda_\beta \left( \frac{m_{cur} - m_{pre}}{m_{pre}} + \frac{p_{cur} - p_{pre}}{p_{pre}} \right) \tag{4}$$

In expression (3) $pre$ denotes the previous iteration's best first stage training candidates, $cur$ denotes the current candidate architectures. $\lambda$ are application-specific constants. And the empirical rule for picking $\lambda$ is to check how much accuracy gain or loss is expected when the user treats the trade-offs problem between the network performance and inference memory consumption.

Table 1: The data lifetime table to calculate the running intermediate memory consumption, 1 denotes the 'alive' state, and 0 denotes the 'dead' or 'sleep' state. The last row calculate the sum of all alive intermediate representation in its time stamp.

| Time | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| $I_1$ | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $I_2$ | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| $I_3$ | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| $I_4$ | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| $I_5$ | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| $I_6$ | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| $I_7$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| $I_8$ | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| $I_9$ | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| $I_{10}$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| #ALIVE | 1 | 4 | 4 | 4 | 5 | 4 | 5 | 3 | 0 |

### 3.4. Lifetime based Approach for Memory Consumption Calculation

In MemNet, the merit of a neural network is evaluated by its memory consumption of the inference process and ultimately the amount of memory hardware that the deployment device has. Therefore, it is important to create a method to estimate these memory related metrics for a given neural network architecture. To do so, we propose a data lifetime based method that can calculate the runtime memory consumption and its upper bound for given neural network architecture.

Lifetime of data is defined as the time from which it is first generated to which it is no longer used. In a feedforward neural network, a computing platform computes each layer's output from the input layer to the output layer. The output data of a layer must be stored in memory until it is used by the subsequent layers. However, once the data is no longer needed, it is discarded and the memory will be used for storing other data.

We can calculate the size of runtime memory consumption by summing all the output data that cannot be discarded at that time. Then the largest memory consumption over the course of all the computations becomes the upper-bound of runtime memory consumption. The platform must have the memory hardware whose size is equal to or greater than the upper bound (plus the total parameter data size).

Here we explain the estimation process using the internal search result network (Figure 3) and the data lifetime table (Table 1). At Time (T) 1, the input ($I_1$) is given and three layers ("Sep 3 $\times$ 3", "Fac 1 $\times$ 7", and "Sep 5 $\times$ 5") use $I_1$ to compute their output. At T=1, therefore, $I_1$ must be stored in memory as three layers use it. But other layers have not produced their outputs, yet, and thus consume

no memory usage. Then, the sum of alive runtime memory consumption at T=1 is 1 since $I_1$ is the only data that need to be stored in memory. Here, for simplicity, we assume all the data size is 1. This process continues to the last layer. After we fill up the table, the lifetime of data is equivalent to the length of the row with non-zero entries. The last row of the table is the sum of alive runtime memory consumption at each time. The largest intermediate representation memory consumption over time, i.e., 5 in this example, represents the upper bound intermediate memory consumption, or memory requirement.

# 4. Experiments

We conduct our experiments on CIFAR-10. It has 50,000 training images and 10,000 test images. We adopt the standard data pre-processing and augmentation techniques, i.e. subtracting the channel mean and dividing the channel standard deviation, centrally padding the training images to 4040 and randomly cropping them back to 32 32, randomly flipping them horizontally and cut-out strategy. After the search is done, we utilize the cell topology to extend to a larger model and train on ImageNet classification task to evaluate the MemNet on large-scale dataset,

For the ralative ranking controller, we use GRU model as the core layer of RNNs, and the hidden state size and embedding size are both 100. All of the controllers are trained using the first and second iterations' result and continuously fine-tuned with the new trained architecture results after each iteration.

During the search, the number of blocks are set to 5, the filter number of each operation layers are set to 64 for CIFAR-10. At each iteration of the search algorithm, K, is set to 100, top of the candidates will be selected to train. For the CNNs architecture training, there are different training strategies for 'augment' and 'trim' candidate architectures. Each 'augment' candidate network is trained for 60 epochs with batch size 128 using Stochastic Gradient Descent with Warm Restarts [20], learning rate of 0.01 and momentum weight of 0.9. And each 'trim' candidate is fine-tuned based on the previous model for 20 epochs with the same batch size and optimization strategy. Specifically, for saving searching time, if the search iteration selects a 'trim' architecture, the top $K$ candidate networks will be trimmed in the same topology and fine-tuned in the next iteration. Therefore, our augment-trim based search process won't desire much more searching time compared to single growth-based methods, which cost around 14 days with 4 GTX 1080 GPUs.

## 4.1. Results on Memory Efficient Architecture Search

**Results on CIFAR-10**. After searching process is done, we show the final architecture with $\lambda_\alpha = 0.5$ and $\lambda_\beta = 0.5$, the best trade-off coefficient from Section 4.2. In this work, we aim to find memory efficient neural architecture with acceptable performance and lower memory consumption. Therefore, our result is compared with three different groups of methods. In the first group, we compare our MemNet with manually designed efficient neural architectures, including ResNet [7] and CondenseNet [11]. We also compare with the single target NAS methods in the second group, which locates the architecture requiring tens of MB total memory consumption, including ENASNet [24], NASNet [37] and [17]. In the end, our MemNet is also compared with other multiple objective NAS methods, including PPP-Net [3] and [21]. In this experiment, we report the network parameters memory usage based on the data from their paper and we will calculate a corresponding intermediate representation memory consumption based on our lifetime based estimation method.

Table 2: The Neural Architecture Search Results on CIFAR-10. 'Params Mem' is the network parameters memory consumption. 'Inters Mem' is the runtime intermediate representation memory consumption. And 'Total Mem' calculate a totally memory consumption. 'Acc' is the top-1 classification acuuracy rate on the CIFAR-10 test set.

| **Model** | **Params Mem** | **Inters Mem** | **Total Mem** | **Acc** |
|---|---|---|---|---|
| ResNet[7] | 6.8MB | 0.42 | 7.22MB | 93.57% |
| CondenseNet[11] | 2.2MB | 5.93MB | 8.73MB | 95% |
| ENASNet[24] | 18.4MB | 7.89MB | 26.29MB | 96.46% |
| NASNet[37] | 13.2MB | 11.8MB | 25MB | 96.59% |
| PNASNet[17] | 12.8MB | 6.2MB | 19MB | 96.37% |
| DVOLER[21] | 2MB | – | – | 95.43% |
| PPPNet[3] | 1.8MB | 5.12MB | 6.92MB | 94.16% |
| **MemNet-90** | 1.76MB | 5.1MB | 6.86MB | 95.57% |
| **MemNet-60** | 0.84MB | 4.98MB | 5.82MB | 94.02% |

As shown in Table 3, the two manually crafted neural architectures, ResNet [7] and CondenseNet [11] indeed achieve a good performance with limited total memory usage. ResNet [7] constructs a very deep network containing 100 layers to pursue a better result. The intermediate representation in this case does not need a lot of memory but parameters do. CondenseNet [11] also constructs a deep network with re-utilization of intermediate representation to pursue a better result, which causes a high intermediate memory consumption. According to the results, we notice that the manual designs can only optimize one of network parameters and intermediate representations at a time. On the other hand, with the automatic design algorithm, second group NAS methods achieve much better performance than the manual design methods. However, without additional optimization target, those networks require more memory

Table 3: 'Params Mem' is the network parameters memory consumption. 'Acc' is the top-1 classification accuracy rate on the CIFAR-10 test set.

| **Model** | **Params Mem** | **Acc** |
|---|---|---|
| NASNet-A[37] | 21.2MB | 73.54% |
| DVOLER-B[21] | 18.92MB | 73.65% |
| PPPNet[3] | 19.2MB | 74.02% |
| **MemNet** | 11.3MB | 73.8% |

consumption, which isn't suitable for mobile devices. In the end, Table 3 also shows the comparative result of our Memnet among the others. Comparing with the DVOLER, our MemNet saves 58% more network parameter consumption only with a loss of 1.4% accuracy. Also, comparing with the PPPNet, our MemNet achieves 17.3% total memory consumption saving and 53% network parameter saving with only a loss of 0.15% accuracy.

**Results on ImageNet**. We further transfer our search architecture to test the performance on ImageNet classification task. The cell typology searched on CIFAR-10 dataset is directly utilized for ImageNet with more repeated blocks and extra filters of each operation layer. The results of ImageNet training is shown in Table 3. Our MemNet reaches a similiar accuracy performance with PPPNet, but saves more than 40% parameter memory usage. Besides, it outperforms DVOLER and NASNet approachers with a higher accuracy performance and also saves more than 40% network parameter memory usage.

## 4.2. Memory Efficient Score Analysis

In this part, we evaluate the efficiency of our memory consumption based score on navigating the search process. We set three experiments to show the capability of our ranking score on navigating the overall search process and selecting memory efficient candidate architectures during the search process.

**Experiments on Memory Efficient Search**. The first experiment is designed to compare our ranking score-based MemNet with a single accuracy-based MemNet. In Figure 6, we show the accuracy, network parameter consumption and intermediate memory consumption of the best candidate architecture in each search iteration. In this experiment, the two methods follow the same MemNet search framework, except that the accuracy-based Memnet utilizes the accuracy as the target to train the controller and select the candidate architectures. The search process on accuracy optimization is stopped after we access an architecture having similar performance with our ranking score-based method. As shown in the left figure in Figure 4, the accuracy-based search barely chose the 'trim' search space and its accuracy in round 6 is similar to the result of our ranking score-based method. At that point, as shown in the middle/right figures of Figure 4, accuracy-based method requires a 200% larger network parameter consumption and 74% larger intermediate memory consumption. In other words, our ranking score shows brilliant capability on selecting a more efficient candidate architecture and utilize the 'trim' generation to cut the redundant connections.

**Experiments on trade-off coefficient**. In the second experiment, we attempt different trade-offs coefficient $\lambda_s = \frac{\lambda_\beta}{\lambda_\alpha}$ between accuracy and memory consumption, shown in Equation 3 and 4. We set 5 different trade-offs coefficients $\frac{\lambda_\beta}{\lambda_\alpha}$ for the loss function of controller and the selection score, including 0:10, 2:8, 5:5, 8:2 and 10:0. In Figure5, as the $\lambda_\alpha$ increases, the searched network requires more memory resources and can reach a higher accuracy. To achieve a network with high performance and memory efficient architecture, the 5:5 coefficient is selected to be the best trades-off solution between the accuracy and memory consumption when calculating the Equation 3 and 4.

**Experiments on Candidate Selection**. In the final experiment, we show the network parameters and intermediate memory consumption of all trained candidate architectures in the first four search rounds. As shown in Figure 6, the vertical axis denotes the network parameter memory consumption and the horizontal axis denotes the intermediate memory consumption. Besides, the gradation of the points represents a related accuracy of these candidates among all of the architectures in its iteration. As can be seen in the Figure 6, the red dotted rectangle bounded region represent the low network parameter candidates architectures and the bottom region represent a low intermediate memory consumption candidates. It's obvious that lots of high accuracy point locate in both top and bottom low parameter regions. Therefore either single accuracy-based method nor network parameters-based methods can discriminate the different usage on intermediate representation, which may lead to select a high total memory consumption architecture. However, considering both network parameters and intermediate representation via lifetime based method, our MemNet focus on the green dotted rectangle bounded region to choose a more efficient candidate architecture.

## 4.3. Ranking Controller Analysis

In the end, we aim to show the importance of the correlation between the candidate architectures extracted by our network-cross controller. Towards that, we adopt a single-layer RNN (GRU) and a double-layer RNN (GRU) as comparison methods. The first one represents the present popular controller structure in NAS to extract the single network's feature for predicting its performance. And the second one does the same thing with the first structure but with a more hidden layer. Both of them will receive the set of block features and predict the corresponding network per-

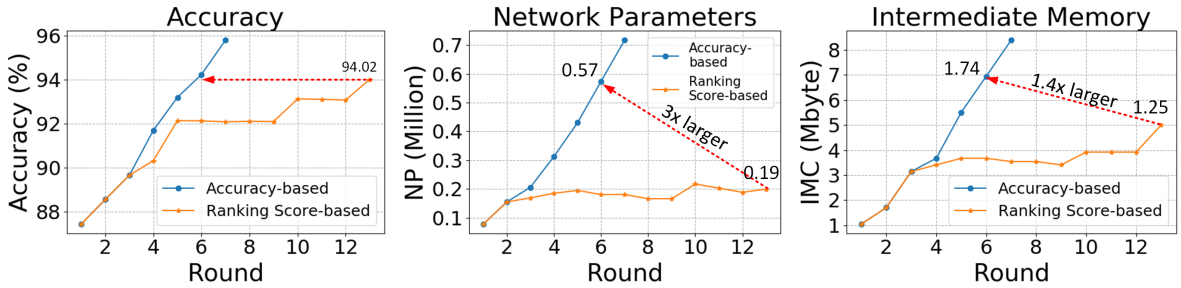


Figure 4: The comparison between accuracy-based and memory efficient score-based MemNet. The dotted line denotes the point having the similar accuracy between those two methods. It shows our ranking score-based MemNet can achieve the same accuracy 94.02% with less network parameter consumption and 0.71 less intermediate representation consumption.

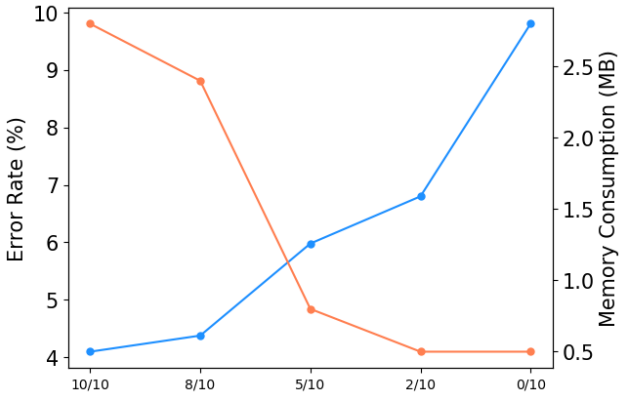


Figure 5: The error rate and memory consumption of search architectures with different loss function coefficient. The points in X-axis represent the $\frac{\lambda_\alpha}{\lambda_\alpha+\lambda_\beta}$.

formance.

Each RNN will be trained by the feature of candidate architectures from $round_0$ with feature padding and predict the ranking results on the candidates of $round_1$. We utilize the Normalized Discounted Cumulative Gain (NDCG) [13] and Average Precision (AP) as the architecture ranking metrics. For both NDCG and AP, we report results at top $K = 50$ and 100 selected by the controller to show the performance of different controller structures. In our experiment, NDCG represents that the importance of the top $K$ selected candidates, which means a higher NDCG denotes the selected top $K$ candidates have higher ranking order in real value. And AP represents that the number of the selected top $K$ candidates in the real top $K$ candidates. As can be seen in Table 4, the double-layer RNN shows high improvement on AP criterion and lightly increment on NDCG criterion, which means those direct prediction methods can extract better feature information with more complicated structure. However, without considering the relation between the candidate architectures, it's hard to obtain more valuable architecture in its prediction results. Differently, our relative ranking RNN extracts correlation information between the candidates to predict a relevant score, which makes a high improvement on both AP and NDCG criterion.

Table 4: The ranking results of different controller structure. 'AP' represent the number of the selected top $K$ candidates in the real top $K$ candidates. 'NDCG' the importance of the selected top $K$ candidates. We show the results on $K$=50 and $K$=100

| **Model** | **AP** | | **NDCG** | |
|---|---|---|---|---|
| | K=50 | K=100 | K=50 | K=100 |
| Single-RNN | 0.066 | 0.237 | 0.043 | 0.078 |
| Double-RNN | 0.128 | 0.268 | 0.062 | 0.080 |
| **RNN(proposed)** | **0.196** | **0.283** | **0.135** | **0.201** |

## 5. Conclusion

The main contribution of this work is to show how we can navigate the NAS framework towards a memory efficient neural architecture design. MemNet, aiming to optimize both network parameter and intermediate memory consumption, utilizes the augment-trim strategy to access more memory efficient candidate architectures. To navigate the search to memory efficient architecture for inference, a customized ranking score is designed by our lifetime based upper bound of memory consumption estimation method. Furthermore, towards the different 'augment' and 'trim' search spaces, we utilize a network-cross controller to extract correlation information to directly predict the ranking results among the untrained candidate architectures. In the end, experimental results on CIFAR-10 demonstrates the effectiveness of MemNet, achieving the competitive accuracy and save an average 24.17% total memory consumption, compared with the state-of-the-art methods.

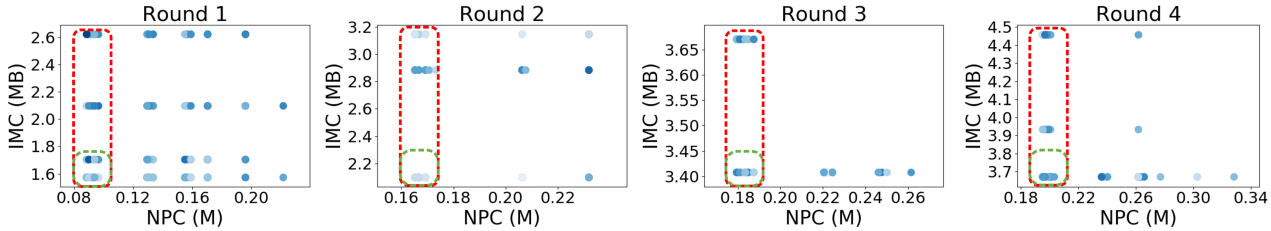


Figure 6: The distribution of memory consumption in the first four search round. Each dot in the figure denotes one trained candidate architecture in each round. The color represents the corresponding accuracy of the candidate. Red area highlight low network parameter consumption architectures and green area highlight both low network parameter and low intermediate memory consumption architectures.

## References


[1] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang. Efficient architecture search by network transformation. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 1

[2] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016. 2

[3] J.-D. Dong, A.-C. Cheng, D.-C. Juan, W. Wei, and M. Sun. Ppp-net: Platform-aware progressive search for pareto-optimal neural architectures, 2018. 1, 2, 6, 7

[4] R. Girshick. Fast r-cnn. *ICCV*, pages 1440–1448, 2015. 1

[5] N. Y. Hammerla, S. Halloran, and T. Plötz. Deep, convolutional, and recurrent models for human activity recognition using wearables. *arXiv preprint arXiv:1604.08880*, 2016. 1

[6] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 2

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CVPR*, pages 770–778, 2016. 1, 6

[8] H. M. Hochman and J. D. Rodgers. Pareto optimal redistribution. *The American economic review*, 59(4):542–557, 1969. 1, 2

[9] 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. 2

[10] C.-H. Hsu, S.-H. Chang, D.-C. Juan, J.-Y. Pan, Y.-T. Chen, W. Wei, and S.-C. Chang. Monas: Multi-objective neural architecture search using reinforcement learning. *arXiv preprint arXiv:1806.10332*, 2018. 1, 2

[11] G. Huang, S. Liu, L. Van der Maaten, and K. Q. Weinberger. Condensenet: An efficient densenet using learned group convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2752–2761, 2018. 2, 6

[12] 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. 2

[13] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002. 8

[14] G. Kortuem, F. Kawsar, V. Sundramoorthy, D. Fitton, et al. Smart objects as building blocks for the internet of things. *IEEE Internet Computing*, 14(1):44–51, 2009. 1

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *NIPS*, pages 1097–1105, 2012. 1, 2

[16] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. 2017. 2

[17] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *ECCV*, pages 19–34, 2018. 3, 6

[18] P. Liu, W. Liu, and H. Ma. Weighted sequence loss based spatial-temporal deep learning framework for human body orientation estimation. *ICME*, pages 97–102, 2017. 1

[19] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016. 1

[20] I. Loshchilov and F. Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016. 6

[21] G. Michel, M. A. Alaoui, A. Lebois, A. Feriani, and M. Felhi. Dvolver: Efficient pareto-optimal neural network architecture search. *arXiv preprint arXiv:1902.01654*, 2019. 1, 2, 6, 7

[22] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Elsevier, 2019. 1

[23] L. Peiye, L. Wu, M. Huadong, M. Tao, and S. Mingoo. Ktan: Knowledge transfer adversarial network. *arXiv preprint arXiv:1810.08126*, 2018. 2

[24] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018. 6

[25] D. Ravi, C. Wong, B. Lo, and G.-Z. Yang. A deep learning approach to on-node sensor data analytics for mobile or

wearable devices. *IEEE journal of biomedical and health informatics*, 21(1):56–64, 2017. 1

[26] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2902–2911. JMLR. org, 2017. 1, 2

[27] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 1

[28] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. *ICLR*, 2014. 2

[29] J. Roschelle. Keynote paper: Unlocking the learning value of wireless mobile devices. *Journal of computer assisted learning*, 19(3):260–272, 2003. 1

[30] M. Sharma, A. Anand, R. Srivastava, and L. Kaligounder. Wearable audio and imu based shot detection in racquet sports. *arXiv preprint arXiv:1805.05456*, 2018. 1

[31] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1

[32] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 2

[33] W. A. Wulf and S. A. McKee. Hitting the memory wall: implications of the obvious. *ACM SIGARCH computer architecture news*, 23(1):20–24, 1995. 1

[34] X. Zhang, Z. Huang, and N. Wang. You only search once: Single shot neural architecture search via direct sparse optimization. *arXiv preprint arXiv:1811.01567*, 2018. 2

[35] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *ICLR*, 2017. 2

[36] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 1, 2

[37] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018. 1, 6, 7