E-DNAS: Differentiable Neural Architecture Search for Embedded Systems

Javier García López
FICOSA ADAS S.L.U
08232 Barcelona, Spain
jgarcia@iri.upc.edu

Antonio Agudo and Francesc Moreno-Noguer
Institut de Robòtica i Informàtica Industrial, CSIC UPC
08028, Barcelona, Spain
{aagudo,fmoreno}@iri.upc.edu

***Abstract*—Designing optimal and light weight networks to fit in resource-limited platforms like mobiles, DSPs or GPUs is a challenging problem with a wide range of interesting applications, *e.g.* in embedded systems for autonomous driving. While most approaches are based on manual hyperparameter tuning, there exist a new line of research, the so-called NAS (Neural Architecture Search) methods, that aim to optimize several metrics during the design process, including memory requirements of the network, number of FLOPs, number of MACs (Multiply-ACcumulate operations) or inference latency. However, while NAS methods have shown very promising results, they are still significantly time and cost consuming.**

In this work we introduce E-DNAS, a differentiable architecture search method, which improves the efficiency of NAS methods in designing light-weight networks for the task of image classification. Concretely, E-DNAS computes, in a differentiable manner, the optimal size of a number of meta-kernels that capture patterns of the input data at different resolutions. We also leverage on the additive property of convolution operations to merge several kernels with different compatible sizes into a single one, reducing thus the number of operations and the time required to estimate the optimal configuration. We evaluate our approach on several datasets to perform classification. We report results in terms of the SoC (System on Chips) metric, typically used in the Texas Instruments TDA2x families for autonomous driving applications. The results show that our approach allows designing low latency architectures significantly faster than state-of-the-art.

***Index Terms*—Deep Learning, Neural Architecture Search, convolutional meta kernels.**

I. Introduction

Designing light Deep Neural Networks (DNNs) and doing it in an efficient manner, are two of the main challenges faced in industries like the automotive, which typically need to deal with resource-constrained platforms. This has been addressed in recent works, like SqueezeNet [1] or MNet [2], focused on optimizing the design of neural networks to alleviate their computational cost without losing performance. Most these studies, however, are based on the optimization of "indirect metrics", such as the number of Multiply-ACcumulate operations (MACs) or the number of architecture parameters, which might not be good approximations to the "direct metrics" like energy consumption or latency. As discussed in [3], [4] or [5], the relationship between these direct and indirect metrics can be highly non-linear and platform-dependent. Another drawback of [1] and [2] is that they require from manual approaches that require expert knowledge, limiting thus their applicability and design efficiency.

The design method has been automatized by the so-called Neural Architecture Search (NAS) [6, 7, 8] approaches. These techniques aim to automatically design light and accurate DNNs by optimizing over a search space defined by all possible operations of the target architecture. This optimization is carried on using either reinforcement learning [6, 7] or evolutionary computing [8].

While NAS-based approaches provide state-of-the-art results in classification tasks for small datasets like CIFAR, they are very computationally and time demanding. There have been attempts to speed up the search process using weight prediction techniques [9] or weight sharing across multiple architectures [10]. Unfortunately, the improvement is still far from providing solutions that can scale to large datasets like ImageNet due to the prohibitive time and resources required.

In this paper we introduce E-DNAS, a differentiable NAS approach that optimizes the direct metrics of an embedded platform, yielding accurate and low-latency DNNs that can be deployed in memory-constrained platforms. The presented research builds upon three main ideas. First, we apply a depth-aware convolution over the input image to compute high-resolution feature maps. Second, we propose a parallel architecture search pipeline that operates on these feature maps and learns the optimal size and parameters of the convolution kernels. This optimization process is ruled by a multi-objective differentiable loss function that combines classification accuracy and minimal latency, a direct metric. And third, we boost the architecture search velocity through a novel block that connects the learned meta-kernels during training. This block is shown in Figure 1 and aims to update the learned meta-kernel (from *feature map 1*) on each iteration with the result of the weighted sum of that kernel and a second one being learned in parallel (the one from *feature map 2*). We show that this training information exchange on each iteration speeds up the search for the optimal kernels.

We demonstrate remarkable results in terms of search-time and classification accuracy compared to other state-of-the-art NAS methods and comparable to other recent breakthroughs like [11] or [12], which are more oriented for mobile devices rather than to be integrated into embedded systems.

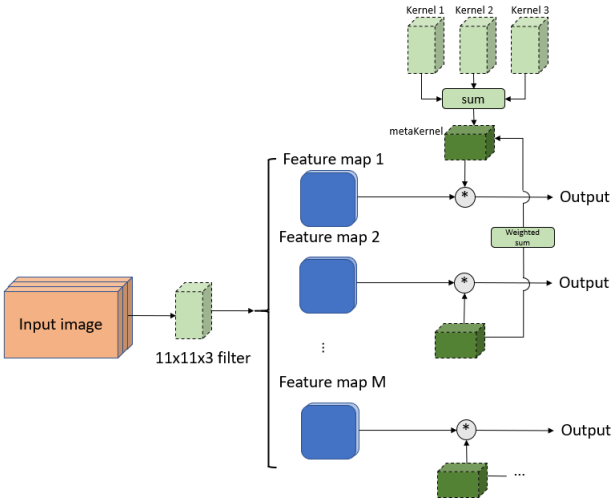

FIG. 1: **General overview of E-DNAS.** Our approach has two main building blocks: a depth-aware convolution with a high resolution 11×11 kernel followed by pairwise learning of meta-kernels with loopy flow of information on each iteration between training paths.

II. Related Work

Despite Deep Learning has changed the panorama of the Artificial Intelligence field, it has some important constraints or limitations that need to be overcome in order to exploit its full potential. Some of these are the large amount of hardware resources (memory, power consumption) needed to run some deep learning applications and also the manual network and parameter configuration traditionally done by experts to obtain an optimal DNN for a particular application.

Along this direction, some works have focused on reducing the network size and optimizing its hyperparameters by pruning weights, like [13] or [14]. Although these approaches could be effective for some applications, manually selecting the redundant weights and using unstructured sparse filters in not so practical for real platforms. Based on a similar idea, recent papers propose a methodology to design networks that can evolve during the design process based on some feedback in order to obtain the optimal number and type of layers for a specific application. These are the so called NAS (neural architecture search) approaches, which automate the architecture design and have shown improved performance compared to the hand-crafted models.

Some NAS approaches like [15], [7] or [6] employ reinforcement learning for finding the best neural architecture [6, 7]. These approaches propose a framework with a recurrent neural network (RNN) as a controller from which child architectures are extracted and trained. Based on the accuracy of these sub-architectures, a reward signal for the controller is calculated and fed back, such that in the next iteration the controller will give higher probabilities to generate architectures with higher accuracies (the controller learns to improve its search over time). Moreover, many of the presented methods aim to search for light and easy-to-integrate DNNs attending to indirect metrics such as MACs. As demonstrated in different works such as [3] or [16], extracting indirect metrics like MACs or number of weights might not be good proxies for the resource consumption of a network and therefore in this paper we attend to a direct metric such as the latency and minimize it during the search process. Although this reward-based approaches showed really good results in providing efficient network architectures to be executed on mobile platforms, they still had one big disadvantage, which is the extremely long training time they require ([6] needs 2000 GPU days in the ImageNet or CIFAR-10 dataset; or the approach proposed in [8] takes 3150 GPU days).

More recently, a faster version of the NAS has been proposed, which can provide an optimal network design quicker by using gradient-based optimizations, like DARTS[11]. The Differentiable Neural Architectural Search (DNAS) approach proposes a relaxation method to transform the search space into a continuous one, such that the architecture can be optimized with respect to its validation set through gradient descent. These techniques achieve a big efficiency improvement reducing drastically the cost of architecture finding in comparison to the non-differentiable approaches (NAS).

Nevertheless, although methods like DARTS have given good results in terms of accuracy and searching time compared to NAS, they still face some weak points, such as the still relatively long time needed for the architecture finding. Together with this, DNAS approaches such as DARTS [11] have proven not to be practical to be used in large datasets.

[fr: No decimos nada de las direct metricas, cuando le hemos dado peso en abstract e intro. Podemos decir que uno de los inconvnientes de estos metodos es que no optimizan diecamente sobre medidads directas? Y acabas diciendo que el metodo que proponemos sí lo hace, lo que le hace mas util a la practica] [jgl: He hecho referencia a las direct metrics un poco más arriba en el párrafo, hablando de que usamos latencia como medida directa en vez de FLOPs (por ejemplo)]

[jgl: He cambiado cómo están puestos los autores en el encabezado porque en el template pone que debe estar cada autor por separado, con el instituto o empresa donde trabajan en cursiva.]

III. Method

We propose a methodology for automatic neural architecture design to be executed on embedded platforms. We demonstrate state-of-the-art results on feature extraction and object classification tasks, as presented in Table I. We present a DNAS approach that aims to find optimal neural architecture with low latency to be executed on memory-constrained System on Chips (SoCs), such as the one we use late in the experimental section.

The pipeline we propose has two main steps:

- High resolution feature extraction through a depth-awarer convolution using large dimension convolutional kernels.
- Pairwise neural architecture cross-search for the calculated feature maps on previous step.

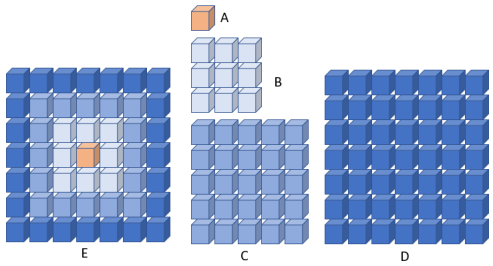

FIG. 2: **Additivity property of convolution:** Convolutions with large kernels can be estimated as a sum of convolutions of small size kernes. In the example, the convolutional kernel (E), results from summing 1x1 kernel (A), 3×3 (B), 5×5 (C) and 7×7 kernel (D).

In this work we regard network latency as the proxy of the computation consumption.

A. Formulation

1) Convolutional filters: As it was demonstrated in AlexNet [17], each convolutional kernel is responsible to capture a local image pattern. The larger the convolutional kernel is, the higher resolution patterns it can detect, although at the cost of more parameters and computations.

On this regard, there is an important idea proposed in the MixConv work [18] that we exploited here and that is, having multiple kernels with different sizes in a single convolution operation can make that our network can capture different types of features from the input images. Based on this, we present a two-step pipeline in which: first, a large convolutional kernel is applied on the input image to capture high resolution patterns, and second, several kernels with different sizes are applied on the calculated feature maps need to be learned during the training process in order to find all different types of patterns present on the input data.

In order to reduce the number of operations and the network size there are two considerations in the above mentioned steps that shall be taken into account:

- The 11×11 filters applied on the feature maps are a sum of 3×3, 5×5 and 7×7 filters applied on the input resource. This work exploits the additive property of convolution: if several 2D kernels with compatible sizes operate on the same input with the same stride to produce outputs of the same resolution, and their outputs are summed up, these kernels are finally added on the corresponding position to obtain an equivalent larger filter that will produce the same output, [19]. See Fig. 2
- The first step proposed in this paper suggests a separable *depth-aware* convolution with a 11×11 kernel that leads to a reduced parameter size and computational cost, compared to the standard convolution operation, [18], [2], [20], [21].

The main difference between the traditional convolution operation and the mentioned separable depth-aware convolution over an input image (or tensor) is the number of steps in which this operation is applied.

In this context, the additivity property is applicable because the sizes of the filter or kernels are compatible, which means, smaller ones can be "contained" in bigger ones (with same center). This property can be formally written as:

$$\boldsymbol{I} * \boldsymbol{K_1} + \boldsymbol{I} * \boldsymbol{K_2} = \boldsymbol{I} * (\boldsymbol{K_1} \oplus \boldsymbol{K_2}), \tag{1}$$

where $\boldsymbol{I}$ is the input feature map, $\boldsymbol{K}_1$ and $\boldsymbol{K}_2$ are two 2D kernels with compatible sizes, and $\oplus$ is the element-wise addition of the kernel or filter parameters on the corresponding positions, [19], [12].

The application of the additivity property is also valid for the following Batch Normalization (BN) so that each single BN applied after each convolution from Eq. (1) produces the same output as the summation of each single convolution and BN with added bias, [19]:

$$\boldsymbol{O} = \boldsymbol{I} * (\frac{\gamma_1}{\sigma_1}\boldsymbol{K_{3\times3}} \oplus \frac{\gamma_2}{\sigma_2}\boldsymbol{K_{3\times1}} \oplus \frac{\gamma_3}{\sigma_3}\boldsymbol{K_{1\times3}}) + b, \tag{2}$$

where $\boldsymbol{O}$ represents the output feature map, $\boldsymbol{I}$ is the input data or feature map generated by the previous layer, σ is batch standard deviation and γ and b are the BN parameters to be learned. $\boldsymbol{I}$ may need to be appropriately padded depending on the resolutions.

2) Feedback-block: One of the contributions of this paper is the addition of one *feedback-block* into the training pipeline of each feature map to update the learned convolutional kernels on each iteration, see Figure 2. The implementation of this *feedback-block* is based on a weighted sum of the learned meta-kernels being trained in parallel:

$$\boldsymbol{K_1}' = \boldsymbol{K_2}' = \beta_1 * \boldsymbol{K_1} + \beta_2 * \boldsymbol{K_2}, \tag{3}$$

where $\boldsymbol{K_1}'$ and $\boldsymbol{K_2}'$ are the two meta-kernels candidates being learned, $\boldsymbol{K_1}$ and $\boldsymbol{K_2}$ are the kernels before the update and β_1 and β_2 are the weights for these kernels. These weights are computed according to the loss on each training "path":

$$\begin{aligned} \beta_1 &= \frac{\tanh\frac{1}{L_1}}{\tanh\frac{1}{L_1} + \tanh\frac{1}{L_2}} \\ \beta_2 &= \frac{\tanh\frac{1}{L_2}}{\tanh\frac{1}{L_1} + \tanh\frac{1}{L_2}} \\ &\text{with } \beta_1 + \beta_2 = 1. \end{aligned} \tag{4}$$

Following Eq. (4) we obtain the weights, which will be closer to one if the calculated loss in the forward pass is small. Through this implementation, on each iteration the closer kernel to the "expected" one has more influence. In Eq. (4) L_1 and L_2 represent the value of the loss function on two parallel network candidates being searched (illustrated in Fig. 1).

After training each image, all learned kernels are encoded into one, following a similar approach as detailed in Eq. (4):

$$\begin{aligned} \boldsymbol{K} &= \sum_{j=1}^{j=N} \Gamma_j * \boldsymbol{K_j} \\ &\text{with } \Gamma_j = \frac{\beta_j}{\sum_i \beta_i}. \end{aligned} \tag{5}$$

B. Search space

We define the search space of each output $x^{(j)}$ (feature map in convolutional networks) as the combination of operations $o^{(i,j)}$ applied on inputs $x^{(i)}$, assuming the inputs as the outputs of the previous two layers, [11]:

$$x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)}). \tag{6}$$

The gradient-based NAS methodologies [11, 22] relax the categorical choice to a softmax to make it continuous. Let O be a set of candidate operations (max pooling, convolutions) where each operation represents some function $o(-)$ to be applied on the input $x^{(i)}$, a particular operation can be represented as:

$$\overline{o}^{(i,j)}(x) = \sum_{o \in O} \frac{\exp({\alpha_o}^{(i,j)})}{\sum_{o' \in O} \exp(\alpha_{o'}{}^{(i,j)})} o(x). \tag{7}$$

As demonstrated in [11], the task of architecture search reduces to learning a set of continuous variables $\alpha = \{\alpha^{(i,j)}\}$.

The relaxation of the categorical choice presented in (7) can also be defined as follows, using the additivity property of convolutions. Based on this, each operation can be calculated as $\boldsymbol{I} * \boldsymbol{K}$, where $\boldsymbol{I}$ is the input of the operation and $\boldsymbol{K}^{(i)}$ is the kernel to be learned:

$$o(x) = \boldsymbol{I} * \boldsymbol{K^{(i)}}. \tag{8}$$

$$\overline{o}^{(i,j)}(x) = \sum_{o \in O} \frac{\exp({\alpha_o}^{(i,j)})}{\sum_{o' \in O} \exp(\alpha_{o'}{}^{(i,j)})} (\boldsymbol{I} * \boldsymbol{K^{(i)}}). \tag{9}$$

After relaxation of the search space, the proposed search network algorithm aims to learn jointly the architecture α and the weights w.

C. Multi-objective loss function

Based on Eq. 9, the goal of the presented DNAS approach shall be to calculate the weights w that minimize the validation loss:

$$\min_{\alpha} = L_{val}(w(\alpha), \alpha). \tag{10}$$

In order to let this method to generate models adaptatively depending on the target embedded platform, we propose to include one more term to the global loss function to be minimized during training that attends to the latency of the network candidate.

The proposed loss function has a term to observe the latency of the proposed architecture. As demonstrated in different works such as [3] or [16], extracting indirect metrics like MACs or number of weights might not be good proxies for the resource consumption of a network since networks with fewer number of MACs can be slower when executed on embedded targets. We define the latency loss as:

$$L_{LAT}(\alpha) = \sum_{l} LAT(b_l)^{(\alpha)}. \tag{11}$$

where $b_l)^{(\alpha)}$ denotes the block at layer-l from the network architecture candidate α [16].

In the presented implementation, we use a latency look up table model to estimate the global latency of the network candidate based on the runtime of each operator, as proposed in [16]. This latency lookup table has been created by checking the runtime of multiple operators on the target platform.

D. The search algorithm

The formulation of the network search problem that is solved through the proposed method can be expressed as follows:

$$\begin{aligned} &\max_{a_i} \text{Accuracy}(a_i) \\ &\text{s.t.} \quad LAT(a_i) <= \text{Budget}, i = 1, ..., N. \end{aligned} \tag{12}$$

where a_i is the sampled network from the search space, $LAT(a_i)$ is the latency on the platform of the sampled network and *Budget* is the predefined latency budget.

The problem presented in equation (12) is solved iteratively by the presented method by minimizing on each iteration the following loss function:

$$L(a, w_a) = CE(a, w_a) + \beta L_{LAT(a)}, \tag{13}$$

where $CE(a, w_a)$ is the cross-entropy loss of the network candidate a with weights w_a and $LAT(a)$ is the measured latency of network candidate a in microseconds, [16].

Typical NAS approaches like [6], [21] or [7] are based on the iteratively training of sampled architectures candidates from the search space on a small proxy dataset through some epochs to be then transferred to the target dataset after training. In the end, the objective of these NAS methodologies is finding the network weights w and optimizing the network candidate $a \in A$ (being A the search space), similar as in the presented work but the needed resources and time to train thousands of network architectures before reaching the optimal solution make them some times infeasible.

Motivated by this problem, DNAS approaches like [11], [22], [16] or [12] have become more popular lately. In this research we adopt a different paradigm of solving the same problem as explained in Section (12) based on DNAS.

In the presented work we relax the categorical choice of a particular filter or kernel in the target architecture by formulating the sampling process in the search stage, similar as proposed in [23] and [16].

$$P(\overline{\boldsymbol{K}} == \boldsymbol{K^i}) = \text{softmax}(\alpha^{(i)}) = \frac{\exp(\alpha^{(i)})}{\sum_{j=0}^{N} \exp(\ \alpha^{(i)}))}. \tag{14}$$

Following this we reformulate Eq. 9 and focus on making Eq. 14 differentiable so that the loss function (13) can be optimized through stochastic gradient descent (SGD) [24], [25], [16].

The objective function (14) is already differentiable with respect to the weights of the kernels but not with respect to the architecture parameters α due to the sampling process. To solve this, we follow a similar approach as in NAS related

Algorithm 1: The search architecture methodology

Result: Find weights w_i and architecture probability parameters α to optimize the global loss function (13), given a defined search space with a combination of operations $\overline{o}^{(i,j)}$, defined in Eq. (9), a latency budget and an input dataset.

random initialization of α parameters

while *not converge* **do**

  Similar to [12], we generate the kernel candidates.

  Calculate Loss through Eq. (13).

  Calculate $\partial L/\partial w_a$ and $\partial L/\partial \alpha$.

  Update weights and architecture probability parameters α .

end

Extract more optimal architecture from learned α parameters.

works [21], [26], [22], [23]. We adopt the Gumbel Softmax function [27] to rewrite the equation (14):

$$P(\overline{\boldsymbol{K}} == \boldsymbol{K^i}) = \frac{\exp((log(\theta_i) + g_i)/\tau)}{\sum_{j=0}^{N} \exp^{(}(log(\theta_i) + g_i)/\tau)}, \qquad (15)$$

where $g_i \sim$ Gumbel(0,1) represents a white noise function that follows the Gumbel distribution between zero and one, τ represents the temperature parameter of the Gumbel Softmax function, [27], which makes the discrete sampling probability function (14) become continuous as τ approximates to one. Lastly θ_i represents the class probabilities calculated in equation 14, [12].

Once the loss function is differentiable, the SGD method to optimize function (13) is applied, so that on each iteration the network architecture weights w_i and probability parameters α are updated based on the partial derivative of the loss function with respect to w and α, respectively.

The search process is now equivalent to training the stochastic network after generating all kernel candidates from the 11×11 meta kernel, similar to [12]. During training, the value of the loss function $L(a, w_a)$ in Eq. (13) is calculated. Together with it, $\partial L/\partial w_a$ and $\partial L/\partial \alpha$ are computed to update weights and architecture probability parameters on each iteration. Through this, we train each operator's weight and update the sampling probability for each operator, respectively, so that when training finishes, we can obtain the optimal network architectures with the best kernels from the learned α parameters.

As it will be shown in the experimental section, the proposed approach works faster than RL and typical NAS methodologies and provides very competitive results for high resolution input images.

IV. Experiments

In this section we aim to demonstrate the performance and efficiency of the proposed method comparing the results obtained on different datasets.

We have conducted experiments on the commonly used ImageNet benchmark [28], KITTI [29] and COCO [30] datasets. We have also applied several important training tricks, which we detail next.

A. Implementation details

We have trained the models using 8 GPU NVIDIA Tesla V100. As proposed in [38] we have applied several implementation tricks to improve the training process, such as the following:

- Randomly sample an image and decode it into 32-bit floating point raw pixel values in [0,255].
- Random crop of a rectangular region.
- Horizontal flip with 0.5 probability.
- Normalize RGB channels.
- Scale hue, saturation and brightness coefficients.

Due to the importance of the learning rate in the training process, in the conducted experiments we have applied a learning rate warm up (use small learning rate at the beginning and then switch back to the initial learning rate when training process is stable) followed by decaying cosine learning rate to improve the training process, as commented in [38] and proposed by [39].

In contrast to the typical exponentially decaying learning rate used $l_r = l_{r0} * e^{-Kt}$ in this work we have applied the following formula:

$$\begin{aligned} l_r &= \frac{1}{2}(1 + \cos\frac{b\pi}{B})l_{r0} \\ w^* &= w + l_r\frac{\partial L}{\partial w}, \end{aligned} \qquad (16)$$

where B is the total number of batches, b is the actual batch during training, w^* the updated weight, w the weight before update and l_{r0} is the initial learning rate (in our case 0.65).

B. Target platform

The embedded platform targeted to check the effectiveness of the proposed method is the TDA2 system on chip which can accelerate deep neural network layers using the C66x DSP cores together with the Texas Instruments Deep Learning suite (TIDL) [40] to convert the trained network from floating point to fixed point and to enable the inference of the network on the embedded platform. The TDA2 hardware has ARM Cortex-A15 cores running at up to 1.5 gigahertz, a dual-core DSP C66x processors that are capable of running deep learning inference, together with one embedded vision engine subsystem (EVE). It can run 16 times 16-bit (enough resolution for deep learning applications) MAC operations per cycle reaching up to 20.8 GMACs/s [41].

To check the effectiveness of the proposed method we have mainly attended to MAC and FLOPs in order to compare the results with the theoretical performance of the target platform. To calculate this, based on the hardware specifications mentioned above the SoC has two C66x DSP cores running at 1000 GHz frequency ($2*32GMACs$), other one EVE core running at 900 MHz ($16*900MMACs$) and other 2 ARM Cortex A15

Model	Search Method	Search Space	Search Dataset	# Params(M)	FLOPs(M)	acc(%)
MNetV2 [31]	manual	-	-	3.4	300	72.0
CondenseNet(G=C=8) [32]	manual	-	-	4.8	529	73.8
EfficientNet-B0 [33]	manual	-	-	5.3	390	76.3
NASNet-A [6]	RL	cell	CIFAR-10	5.3	564	74.0
PNASNet [34]	SMBO	cell	CIFAR-10	5.1	588	74.2
DARTS [11]	gradient	cell	CIFAR-10	4.7	574	73.3
PDARTS [35]	gradients	cell	CIFAR-10	4.9	557	75.6
GDAS [23]	gradients	cell	CIFAR-10	4.4	497	72.5
MnasNet [21]	RL	stage-wise	ImageNet	3.9	312	75.2
Single-Path NAS [36]	gradients	layer-wise	ImageNet	4.3	365	75.0
ProxylessNAS-R [26]	RL	layer-wise	ImageNet	4.1	320	74.6
ProxylessNAS-G [26]	gradient	layer-wise	ImageNet	-	-	74.2
FBNet [16]	gradients	layer-wise	ImageNet	5.5	375	74.9
MNetV3 Large [37]	RL	layer-wise	ImageNet	5.4	219	75.2
MNetV3 Small [37]	RL	layer-wise	ImageNet	2.9	66	67.4
MixNet [18]	RL	kernel-wise	ImageNet	5.0	360	77.0
MetaKernels [12]	gradient	kernel-wise	ImageNet	7.2	357	77.0
Ours	**RL**	**parallel kernel-wise**	**ImageNet**	**5.9**	**365**	**76.9**

TABLE I: ImageNet classification performance compared with other state-of-the-art methods. The proposed approach in this paper demonstrates a good Top-1 accuracy with less number of parameters and FLOPs. The number of parameters, FLOPs and Top-1 accuracy metrics presented in this table for the rest of the methodologies have been directly extracted from their respective papers.

Model	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	bike	person	plant	sheep	sofa
MNetV2 [31]	75.8	84.5	83.4	76.1	68.3	58.7	78.9	84.8	86.5	54.4	80.7	70.9	84.0	85.0	83.6	76.8	48.7	78.7	72.8
MNetV3-S [37]	69.3	77.4	76.9	67.0	62.0	43.7	76.3	79.1	82.1	47.2	75.4	65.2	78.4	81.0	79.7	72.5	39.4	68.7	67.1
MNetV3-L	76.7	84.1	84.1	77.0	69.9	75.9	84.8	85.1	88.1	56.3	84.8	64.8	84.3	87.9	84.7	77.9	46.3	80.5	73.9
metaKernel [12]	77.3	86.1	84.8	76.8	68.6	59.2	83.6	86.3	87.1	56.9	85.2	67.2	86.6	87.2	86.0	77.7	49.1	80.8	74.5
Ours	**77.83**	**85.86**	**84.6**	**74.9**	**70.12**	**57.46**	**63.4**	**87.8**	**88.12**	**58.71**	**83.98**	**72.09**	**86.23**	**86.36**	**86.78**	**87.1**	**50.66**	**79.57**	**74.98**
Ours-A (KITTI)	**77.45**	**-**	**86.11**	**-**	**-**	**-**	**71.8**	**82.67**	**79.35***	**-**	**79.35***	**-**	**79.35***	**79.35***	**81.1**	**86.85**	**-**	**-**	**-**
Ours-B (KITTI)	**77.61**	**-**	**86.34**	**-**	**-**	**-**	**71.8**	**82.92**	**79.42***	**-**	**79.42***	**-**	**79.42***	**79.42***	**80.03**	**86.27**	**-**	**-**	**-**

TABLE II: Comparison of the obtained results on the Pascal VOC2007 test set. In the last two rows are the detection results with KITTI dataset [29] presented using two different configurations of the presented pipeline: in "A" we use a first convolutional kernel of 11×11 (as detailed in Section III) and in "B" we increase the size of the initial kernel to 17×17 to compensate the big resolution images from the KITTI dataset (1382×512 px.). In both cases results are similar although number of operations and therefore cost search in case "B" is bigger (stride had to be reduced in case B to avoid losing features on the first convolutional step). * For the detections in KITTI all types of animals were clustered in a single "animal" class.

cores running at 1500 MHz ($2{*}8{*}1500 MACs$). This results in 105 GMACs as theoretical performance, which means 210 GDLOPs, assuming DLOPs as 8-bit arithmetic or conditional operation (Multiply/Add/Compare). It can be assumed than $1MAC = 2DLOPS$.

For the experiments done on the mentioned TI and presented in Table III, the trained networks using the proposed method in this paper and ImageNet benchmark were converted from floating-point to fixed point to be then executed on the DSP dual core of the above mentioned hardware. In the floating-point to fixed-point conversion an accuracy loss of around 3 $\%$ could be extracted from the results.

C. Results

The experimental results and comparison with other state-of-the-art methods are summarized in Table I.It can be ween that our method achieves a good Top-1 accuracy better than several methods, despite we use a lower number of parameters and FLOPs.

For the first variant of this method in which input data passes first through a convolution with 11×11 kernel we find that a better accuracy with a slight increment of the number of FLOPs can be achieved by increasing the size of the first filter to 13×13 and up to 17×17. Beyond that, the rate between accuracy and number of parameters decreases.

The motivation behind the usage of the first convolution with a 11×11 kernel is because a high resolution kernel that would capture high resolution patterns on the image and reduce its size and with compatible size with the following meta-kernel sizes was needed. Together with this, after several tests, like in Table II, we have empirically seen that the mentioned 11×11 kernel size gives the best trade-off between accuracy, number of operations and simplicity in the implementation. Larger kernel sizes increase the model size with more parameters and also more operations and for this reason, using bigger kernels in the initial step of the pipeline would have led to a bigger network, not so suitable for embedded targets.

With regard to the search process speed, the experiments show that our proposal achieves an optimal architecture faster than other DNAS works, as it can be seen in Figure 3. Our experiment results are summarized in Table 3 where we compare our method with state-of-the-art efficient models both designed automatically and manually. In the case of the MnasNet [21], this paper does not disclose the exact search cost (in terms of GPU-hours or days) so in this paper we have assumed the prediction made for the search cost in MnasNet by [26] and [12].

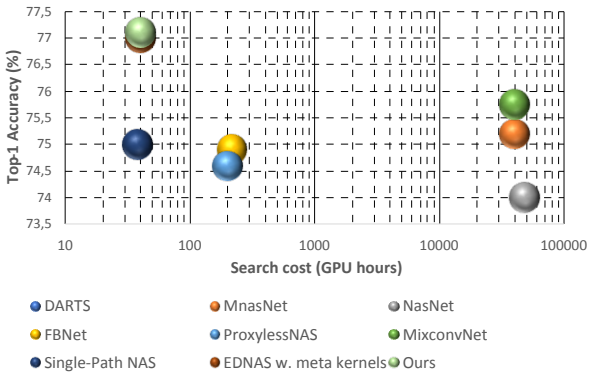

FIG. 3: Comparison of several NAS and DNAS methods in terms of their search cost. The data for the search cost of these works has been directly obtained from their papers. As also commented in [16], the search cost for MnasNet is estimated according to the description in [21]. The search cost of PNAS [34] is estimated based on the results claimed on that work that their method is 8x faster than NAS [6].

Model	# Params (M)	MACs (M)	Time (ms)
MNet [2]	4.2	569	75
NasNet-A [6]	5.3	564	183
Ours	**5.9**	**535**	**38**

TABLE III: Results on ImageNet Benchmark comparing extracted multiply-accumulate operations from different methods and ours. The estimated inference latency on the described TI platform based on the calculated MACs is 38ms.

V. CONCLUSION

In this work we present a network search approach to design light and optimal DNNs reducing the searching time. We propose a two-step pipeline that learns different meta-kernel sizes, able to treat different resolution patterns. We propose a pairwise searching with circular feedback on each iteration to speed up the process by updating the target weights and network parameters iteratively with, not only the loss calculated during its training, but also with the loss calculated on the parallel kernel being learned.

We demonstrate that our method provides good results in terms of accuracy and searching speed compared to other methods like [21] or [11].

REFERENCES

[1] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, Kurt Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size," *International Conference on Learning Representations (ICLR)*, 2017.

[2] 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*, vol. abs/1704.04861, 2017.

[3] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, Hartwig Adam, "Netadapt: Platform-aware neural network adaptation for mobile applications," *European Conference on Computer Vision (ECCV)*, 2018.

[4] Liangzhen Lai, Naveen Suda, Vikas Chandra, "Not all ops are created equal!" *SysML*, 2018.

[5] Tien-Ju Yang, Yu-Hsin Chen, Vivienne Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[6] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, Quoc V. Le, "Learning transferable architectures for scalable image recognition," *International conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[7] Barret Zoph, Quoc V. Le, "Neural architecture search with reinforcement learning," *International Conference on Learning Representation (ICLR)*, 2016.

[8] Esteban Real, Alok Aggarwal, Yanping Huang, Quoc V Le, "Regularized evolution for image classifier architecture search," *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[9] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston., "Smash: one-shot model architecture search through hypernetworks," *International Conference on Learning Representations (ICLR)*, 2018.

[10] Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter., "Simple and efficient architecture search for convolutional neural networks." *International Conference on Learning Representations (ICLR)*, 2018.

[11] Hanxiao Liu, Karen Simonyan, Yiming Yang, "Darts: Differentiable architecture search," *International Conference on Learning Representation (ICLR)*, 2019.

[12] Shoufa Chen, Yunpeng Chen, Shuicheng Yan, Jiashi Feng, "Efficient differentiable neural architecture search with meta kernels," *Computer Vision and Pattern Recognition (CVPR)*, 2019.

[13] Song Han, Jeff Pool, John Tran, William J. Dally, "Learning both weights and connections for efficient neural networks," *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[14] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, Jan Kautz, "Pruning convolutional neural networks for resource efficient inference," *International Conference on Learning Representation (ICLR)*, 2017.

[15] Y. He and S. Han, "Adc: Automated deep compression and acceleration with reinforcement learning," *ArXiv*, vol. abs/1802.03494, 2018.

[16] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun,Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," *Computer Vision and Pattern Recognition (CVPR)*, 2019.

[17] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, "Imagenet classification with deep convolutional neural networks," 2015. [Online]. Available: http://vision.stanford.edu/teaching/cs231b_spring1415/slides/alexnet_tugce_kyunghee.pdf

[18] Mingxing Tan, Quoc V. Le, "Mixconv: Mixed depthwise convolutional kernels," *British Machine Vision Conference (BMCV)*, 2019.

[19] Xiaohan Ding, Yuchen Guo, Guiguang Ding, Jungong Han, "Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks," *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019.

[20] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, Jian Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," *European Conference on Computer Vision (ECCV)*, 2018.

[21] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, Quoc V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," *Computer Vision and Pattern Recognition (CVPR)*, 2019.

[22] Guilin Li, Xing Zhang, Zitong Wang, Zhenguo Li, Tong Zhang, "Stacnas: Towards stable and consistent differentiable neural architecture search," *Computer Vision and Pattern Recognition (CVPR)*, 2019.

[23] Xuanyi Dong, Yi Yang, "Searching for a robust neural architecture in four gpu hours," *Computer Vision and Pattern Recognition (CVPR)*, 2019.

[24] Cui, Xiaodong and Zhang, Wei and Tüske, Zoltán and Picheny, Michael, "Evolutionary stochastic gradient descent for optimization of deep neural networks," *Advances in Neural Information Processing Systems (NIPS)*, 2018.

[25] Sebastian Ruder, "An overview of gradient descent optimizationalgorithms," 2017. [Online]. Available: https://arxiv.org/pdf/1609.04747.pdf

[26] Han Cai, Ligeng Zhu, Song Han, "Proxyless nas: Direct neural architecture search on target task and hardware," *International Conference on Learning Representation (ICLR)*, 2019.

[27] Eric Jang, Shixiang Gu, Ben Poole, "Categorical reparameterization with gumbel-softmax," *International Conference on Learning Representation (ICLR)*, 2017.

[28] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," *Computer Vision and Pattern Recognition (CVPR)*, 2009.

[29] Andreas Geiger, Philip Lenz, Christoph Stiller and Raquel Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research (IJRR)*, 2013.

[30] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. L. Zitnick, "Microsoft coco: Common objects in context," *European Conference for Computer Vision (ECCV)*, 2014.

[31] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," *Computer Vision and Pattern Recognition (CVPR)*, 2019.

[32] Gao Huang, Shichen Liu, Laurens van der Maaten, Kilian Q. Weinberger, "Condensenet: An efficient densenet using learned group convolutions," *Computer Vision and Pattern Recognition (CVPR)*, 2018.

[33] Mingxing Tan, Quoc V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *International Conference on Machine Learning (ICML)*, 2019.

[34] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, Kevin Murphy, "Progressive neural architecture search," *European Conference Computer Vision (ECCV)*, 2018.

[35] Xin Chen, Lingxi Xie, Jun Wu, Qi Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," 2019. [Online]. Available: https://arxiv.org/abs/1904.12760

[36] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, Diana Marculescu, "Single-path nas: Designing hardware-efficient convnets in less than 4 hours," 2019. [Online]. Available: https://arxiv.org/abs/1904.02877

[37] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, Hartwig Adam, "Searching for mobilenetv3," *International Conference on Computer Vision (ICCV)*, 2019.

[38] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, Mu Li, "Bag of tricks for image classification with convolutional neural networks," *Computer Vision and Pattern Recognition (CVPR)*, 2018.

[39] TIlya Loshchilov, Frank Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *International Conference on Learning Representations (ICLR)*, 2017.

[40] TEXAS INSTRUMENTS manual, "Texas instruments technical manual," 2018. [Online]. Available: http://www.ti.com/lit/wp/spry314/spry314.pdf

[41] TEXAS INSTRUMENTS, "Texas instruments tidl," 2018. [Online]. Available: http://www.ti.com/tool/SITARA-MACHINE-LEARNING