LightTrack: Finding Lightweight Neural Networks for Object Tracking via One-Shot Architecture Search

Bin Yan[1,2,*], Houwen Peng[1,*,†], Kan Wu[1,3,*], Dong Wang[2], Jianlong Fu[1], and Huchuan Lu[2,4]
[1]Microsoft Research Asia [2]Dalian University of Technology
[3]Sun Yat-sen University [4]Peng Cheng Laboratory

Abstract

Object tracking has achieved significant progress over the past few years. However, state-of-the-art trackers become increasingly heavy and expensive, which limits their deployments in resource-constrained applications. In this work, we present LightTrack, which uses neural architecture search (NAS) to design more lightweight and efficient object trackers. Comprehensive experiments show that our LightTrack is effective. It can find trackers that achieve superior performance compared to handcrafted SOTA trackers, such as SiamRPN++ [30] and Ocean [56], while using much fewer model Flops and parameters. Moreover, when deployed on resource-constrained mobile chipsets, the discovered trackers run much faster. For example, on Snapdragon 845 Adreno GPU, LightTrack runs 12× faster than Ocean, while using 13× fewer parameters and 38× fewer Flops. Such improvements might narrow the gap between academic models and industrial deployments in object tracking task. LightTrack is released at here.

1. Introduction

Object tracking is one of the most fundamental yet challenging tasks in computer vision. Over the past few years, due to the rise of deep neural networks, object tracking has achieved remarkable progress. But meanwhile, tracking models are becoming increasingly heavy and expensive. For instance, the latest SiamRPN++ [30] and Ocean [56] trackers respectively utilize 7.1G and 20.3G model Flops as well as 11.2M and 25.9M parameters to achieve state-of-the-art performance, being much more complex than the early SiamFC [5] method (using 2.7G Flops and 2.3M parameters), as visualized in Fig. 1. Such large model sizes and expensive computation costs hinder the deployment of tracking models in real-world applications, such as camera drones, industrial robotics, and driving assistant system, where model size and efficiency are highly constrained.

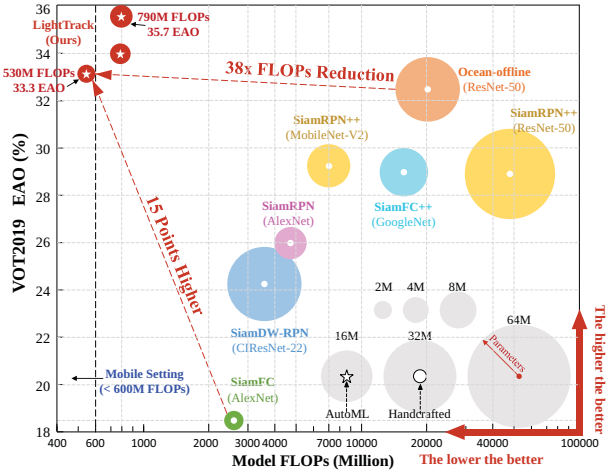

Figure 1: Comparisons with state-of-the-art trackers in terms of EAO performance, model Flops and parameters on VOT-19 benchmark. The circle diameter is in proportion to the size of model parameter. The proposed LightTrack is superior than SiamFC [5], SiamRPN [31], SiamRPN++ [30], SiamFC++ [52] and Ocean [56], while using much fewer Flops and parameters. Best viewed in color.

There are two straightforward ways to tackle the complexity and efficiency issues. One is model compression, while the other is compact model designing. Existing off-the-shelf compression techniques such as pruning and quantization can reduce model complexity, while they inevitably bring non-negligible performance degradation due to information loss [21, 38]. On the other hand, handcrafting new compact and efficient models is engineering expensive and heavily relies on human expertise and experience [55, 15].

This paper introduces a new solution – automating the design of lightweight models with *neural architecture search* (NAS), such that the searched trackers can be carried out in an efficient fashion on resource-limited hardware platforms. It is non-trivial because that object trackers typically need ImageNet pre-training, while NAS algorithms require the performance feedback on the target tracking task as supervision signals. Based upon recent one-

*Equal contributions. Work performed when Bin and Kan are interns of MSRA. † Corresponding author: houwen.peng@microsoft.com.

shot NAS [41, 4, 20], we propose a new search algorithm dedicated to object tracking task, called *LightTrack*. It encodes all possible architectures into a backbone supernet and a head supernet. The backbone supernet is pre-trained on ImageNet then fine-tuned with tracking data, while the head supernet is directly trained on tracking data. The supernets are trained only once, then each candidate architecture inherits its weights from the supernets directly. Architecture search is performed on the trained supernets, using tracking accuracy and model complexity as the supervision guidance. On the other hand, to reduce model complexity, we design a search space consisting of lightweight building blocks, such as depthwise separable convolutions [11] and inverted residual structure [45, 23]. Such search space allows the one-shot NAS algorithm to search for more compact neural architectures, striking a balance between tracking performance and computational costs.

Comprehensive experiments verify that LightTrack is effective. It is able to search out efficient and lightweight object trackers. For instance, LightTrack finds a 530M Flops tracker, which achieves an EAO of 0.33 on VOT-19 benchmark, surpassing the SOTA SiamRPN++ [30] by 4.6% while reducing its model complexity (48.9G Flops) by 98.9%. More importantly, when deployed on resource-limited chipsets, such as edge GPU and DSP, the discovered tracker performs very competitive and runs much faster than existing methods. On Snapdragon 845 Adreno 630 GPU [3], our LightTrack runs 12× faster than Ocean [56] (38.4 *v.s.* 3.2 *fps*), while using 13× fewer parameters (1.97 *v.s.* 25.9 M) and 38× fewer Flops (530 *v.s.* 20,300 M). Such improvements enable deep tracking models to be easily deployed and run at real-time speed on resource-constrained hardware platforms.

This work makes the following contributions.

- We present the first effort on automating the design of neural architectures for object tracking. We develop a new formulation of one-shot NAS and use it to find promising architectures for tracking.
- We design a lightweight search space and a dedicated search pipeline for object tracking. Experiments verify the proposed method is effective. Besides, the searched trackers achieve state-of-the-art performance and can be deployed on diverse resource-limited platforms.

2. Related Work

Object Tracking. In recent years, siamese trackers have become popular in object tracking. The pioneering works are SiamFC and SINT [5, 47], which propose to combine naive feature correspondence with the siamese framework. A large number of follow-up works have been proposed and achieved significant improvements [10, 18, 32, 34, 49]. They mainly fall into three camps: more precise box estimation, more powerful backbone, and online update. More concretely, in contrast to the multiple-scale estimation in SiamFC, later works like SiamRPN [31] and SiamFC++ [52] leverage either anchor-based or anchor-free mechanism for bounding box estimation, which largely improve the localization precision. Meanwhile, SiamRPN++ [30] and Ocean [56] take the powerful ResNet-50 [22] instead of AlexNet [29] as the backbone to enhance feature representation capability. On the other hand, ATOM [14], DiMP [6], and ROAM [53] combine online update [40] with the siamese structure and achieve state-of-the-art performance.

Though these methods achieve remarkable improvements, yet they bring much additional computation workload and large memory footprint, thus limiting their usage in real-world applications. For example, deep learning on mobile devices commonly requires model Flops to be less than 600M Flops [7], *i.e.*, *mobile setting*. However, SiamRPN++ [30] with ResNet-50 backbone has 48.9G Flops, which exceeds the mobile setting by ∼80 times. Even SiamFC [5], using the shallow AlexNet, still cannot satisfy the restricted computation workload when deployed on embedded devices. In summary, there is a lack of studies on finding a good trade-off between model accuracy and complexity in object tracking.

Neural Architecture Search. NAS aims at automating the design of neural network architectures. Early methods search a network using either reinforcement learning [58] or evolution algorithms [51]. These approaches require training thousands of architecture candidates from scratch, leading to unaffordable computation overhead. Most recent works resort to the one-shot weight sharing strategy to amortize the searching cost [33, 41]. The key idea is to train a single over-parameterized hypernetwork model, and then share the weights across subnets. Single-path with uniform sampling [20] is one representative method in one-shot regime. In each iteration, it only samples one random path and trains the path using one batch data. Once the training process is finished, the subnets can be ranked by the shared weights. On the other hand, instead of searching over a discrete set of architecture candidates, differentiable methods [37, 8] relax the search space to be continuous, such that the search can be optimized by the efficient gradient descent. Recent surveys on NAS can be found in [15].

NAS is primarily proposed for image classification and recently extended to other vision tasks, such as image segmentation [36] and object detection [19]. Our work is inspired by the recent DetNAS [9], but has three fundamental differences. First, the studied task is different. DetNAS is designed for object detection, while our work is for object tracking. Second, DetNAS only searches for backbone networks by fixing the head network with a pre-defined handcrafted structure. This may lead to that the searched

backbone is sub-optimal, because it is biased towards fitting the fixed head, rather than the target task. In contrast, our method searches backbone and head architectures simultaneously, aiming to find the most promising combination for the target tracking task. Last, the search space is different. We design a new search space for object tracking dedicated to search for lightweight architectures.

3. Preliminaries on One-Shot NAS

Before introducing the proposed method, we briefly review the one-shot NAS approach, which serves as the basic search algorithm discussed in this work. One-shot NAS treats all candidate architectures as different subnets of a supernet and shares weights between architectures that have common components. More concretely, the architecture search space $\mathcal{A}$ is encoded in a supernet, denoted as $\mathcal{N}(\mathcal{A}, W)$, where W is the weight of the supernet. The weight W is shared across all the architecture candidates, *i.e.*, subnets $\alpha \in \mathcal{A}$ in $\mathcal{N}$. The search of the optimal architecture α^* is formulated as a nested optimization problem:

$$\begin{aligned} \alpha^* &= \arg\max_{\alpha \in \mathcal{A}} Acc_{val}\left(\mathcal{N}(\alpha, W^*(\alpha))\right), \\ \text{s.t.} \quad W^* &= \arg\min_{W} \mathcal{L}_{train}(\mathcal{N}(\mathcal{A}, W)), \end{aligned} \tag{1}$$

where the constraint function is to optimize the weight W of the supernet $\mathcal{N}$ by minimizing the loss function $\mathcal{L}_{train}$ on *training* dataset, while the objective function is to search architectures via ranking the accuracy Acc_{val} of subnets on *validation* dataset based on the learned supernet weight W^*. Only the weights of the single supernet $\mathcal{N}$ need to be trained, and subnets can then be evaluated without any separate training by inheriting trained weights from the one-shot supernet. This greatly speeds up performance estimation of architectures, since no subnet training is required, resulting in the method only costs a few GPU days.

To reduce memory footprint, one-shot methods usually sample subnets from the supernet $\mathcal{N}$ for optimization. For simplicity, this work adopts the single-path uniform sampling strategy, *i.e.*, each batch only sampling one random path from the supernet for training [33, 20]. This single-path one-shot method decouples the supernet training and architecture optimization. Since it is impossible to enumerate all the architectures $\alpha \in \mathcal{A}$ for performance evaluation, we resort to evolutionary algorithms [42, 20] to find the most promising subnet from the one-shot supernet.

4. LightTrack

Searching lightweight architectures for object tacking is a non-trivial task. There exist three key challenges.

- First, in general, object trackers need model pre-training on image classification task for a good initialization, while NAS algorithms require supervision signals from target tasks. Searching architectures for object tracking requires to consider both the pre-training on ImageNet and the fine-tuning on tracking data.
- Second, object trackers usually contain two parts: a backbone network for feature extraction and a head network for object localization. When searching for new architectures, NAS algorithms needs to consider the two parts as a whole, such that the discovered structures are suitable for the target tracking task.
- Last but not the least, search space is critical for NAS algorithms and it defines which neural architectures a NAS approach might discover in principle. To find lightweight architectures, the search space requires to include compact and low-latency building blocks.

In this section, we tackle the aforementioned challenges and propose LightTrack based on one-shot NAS. We first introduce a new formulation of one-shot NAS specialized for object tracking task. Then, we design a lightweight search space consisting of depthwise separable convolutions [11] and inverted residual structure [45, 23], which allows the construction of efficient tracking architectures. At last, we present the pipeline of LightTrack, which is able to search diverse models for different deployment scenarios.

4.1. Tracking via One-Shot NAS

Current prevailing object trackers (such as [31, 14, 6]) all require ImageNet pre-training for their backbone networks, such that the trackers can obtain good image representation. However, for architecture search, it is impossible to pre-train all backbone candidates individually on ImageNet, because the computation cost is very huge (ImageNet pre-training usually takes several days on 8 V100 GPUs just for a single network). Inspired by one-shot NAS, we introduce the weight-sharing strategy to eschew pre-training each candidate from scratch. More specifically, we encode the search space of backbone architectures into a supernet $\mathcal{N}_b$. This backbone supernet only needs to be pre-trained once on ImageNet, and its weights are then shared across different backbone architectures which are subnets of the one-shot model. The ImageNet pre-training is performed by optimizing the classification loss function $\mathcal{L}^{cls}_{pre\text{-}train}$ as

$$W_b^p = \arg\min_{W_b} \mathcal{L}^{cls}_{pre\text{-}train}(\mathcal{N}_b(\mathcal{A}_b, W_b)), \tag{2}$$

where $\mathcal{A}_b$ represents the search space for backbone architectures, while W_b denotes the parameter of the backbone supernet $\mathcal{N}_b$. The pre-trained weight W_b^p are shared across different backbone architectures and serve as the initialization for the subsequent search of tracking architectures. Such weight-sharing scheme allows the ImageNet pre-training to be performed only on the backbone supernet

instead of each subnet, thereby reducing the training costs by orders of magnitude.

Deep neural networks for object tracking generally contain two parts: one pre-trained backbone network for feature extraction and one head network for object localization. These two parts work together to determine the capacity of a tracking architecture. Therefore, for architecture search, it is critical to search the backbone and head networks as a whole, such that the discovered structure is well-suited to tracking task. To this end, we construct a tracking supernet $\mathcal{N}$ consisting of the backbone part $\mathcal{N}_b$ and the head part $\mathcal{N}_h$, which is formulated as $\mathcal{N} = \{\mathcal{N}_b, \mathcal{N}_h\}$. The backbone supernet $\mathcal{N}_b$ is first pre-trained on ImageNet by Eq. (2) and generates the weight W_b^p. The head supernet $\mathcal{N}_h$ subsumes all possible localization networks in the space $\mathcal{A}_h$ and shares the weight W_b across architectures. The joint search of backbone and head architectures is conducted on *tracking* data, which reformulates the one-shot NAS as

$$\begin{aligned} &\alpha_b^*, \alpha_h^* = \underset{\alpha_b, \alpha_h \in \mathcal{A}}{\arg\max}\, Acc_{val}^{trk}\left(\mathcal{N}(\alpha_b, W_b^*(\alpha_b); \alpha_h, W_h^*(\alpha_h))\right), \\ &\text{s.t.} \quad W_b^*, W_h^* = \underset{W_b \leftarrow W_b^p, W_h}{\arg\min}\, \mathcal{L}_{train}^{trk}(\mathcal{N}(\mathcal{A}_b, W_b; \mathcal{A}_h, W_h)), \end{aligned} \tag{3}$$

where the constraint function is to train the tracking supernet $\mathcal{N}$ and optimize the weights W_b and W_h simultaneously, while the objective function is to find the optimal backbone α_b^* and the head α_h^* via ranking the accuracy Acc_{val}^{trk} of candidate architectures on *validation* set of the tracking data. The evaluation of Acc_{val}^{trk} only requires inference because the weights of the architectures α_b and α_h are inherited from $W_b^*(\alpha_b)$ and $W_h^*(\alpha_h)$ (without the need of extra training). Note that, before starting the supernet training, we use the pre-trained weight W_b^p to initialize the parameter W_b, *i.e.*, $W_b \leftarrow W_b^p$, which speeds up convergence while improving tracking performance. During search, it is unaffordable to rank the accuracy of all the architectures in search space, the same as previous work [20, 9], we resort to evolutionary algorithms [42, 20] to find the most promising one.

Architecture Constraints. In real-world deployments, object trackers are usually required to satisfy additional constraints, such as memory footprint, model Flops, energy consumption, etc. In our method, we mainly consider the model size and Flops, which are two key indicators when evaluating whether a tracker can be deployed on specific resource-constrained devices. We preset budgets on networks' *Params* and *Flops* and impose constraints as

$$\begin{aligned} Flops(\alpha_b^*) + Flops(\alpha_h^*) &\leq Flops_{max}, \\ Params(\alpha_b^*) + Params(\alpha_h^*) &\leq Params_{max}. \end{aligned} \tag{4}$$

The evolutionary algorithm is flexible in dealing with different budget constraints, because the mutation and crossover processes can be directly controlled to generate proper candidates to satisfy the constraints [20]. Search can also be repeated many times on the same supernet once trained, using different constraints (e.g., $Flops_{max}$ = 600M or others). These properties naturally make one-shot paradigm practical and effective for searching tracking architectures specialized to diverse deployment scenarios.

Table 1: Search space and supernet structure. "$N_{choices}$" represents the number of choices for the current block. "Chn" and "Rpt" denote the number of channels per block and the maximum number of repeated blocks in a group, respectively. "Stride" indicates the convolutional stride of the first block in each repeated group. The classification and regression heads are allowed to use different numbers of channels, denoted as $C_1, C_2 \in \{128, 192, 256\}$. The input is a search image with size of 256×256×3.

	Input Shape	Operators	$N_{choices}$	Chn	Rpt	Stride
Backbone	$256^2 \times 3$	3×3 Conv	1	16	1	2
	$128^2 \times 16$	DSConv	1	16	1	1
	$128^2 \times 16$	MBConv	6	24	2	2
	$64^2 \times 24$	MBConv	6	40	4	2
	$32^2 \times 40$	MBConv	6	80	4	2
	$16^2 \times 80$	MBConv	6	96	4	1
Cls Head	$16^2 \times 128$	DSConv	6	C_1	1	1
	$16^2 \times C_1$	DSConv / Skip	3	C_1	7	1
	$16^2 \times C_1$	3x3 Conv	1	1	1	1
Reg Head	$16^2 \times 128$	DSConv	6	C_2	1	1
	$16^2 \times C_2$	DSConv / Skip	3	C_2	7	1
	$16^2 \times C_2$	3x3 Conv	1	4	1	1

4.2. Search Space

To search for efficient neural architectures, we use depthwise separable convolutions (DSConv) [11] and mobile inverted bottleneck (MBConv) [45] with squeeze-excitation module [24, 23] to construct a new search space. The space is composed of a backbone part $\mathcal{A}_b$ and a head part $\mathcal{A}_h$, which are elaborated in Tab. 1.

Backbone Space $\mathcal{A}_b$. There are six basic building blocks in the backbone space, including MBConv with kernel sizes of $\{3, 5, 7\}$ and expansion rates of $\{4, 6\}$. Backbone candidates are constructed by stacking the basic blocks. All candidates in the space have 4 stages with a total stride of 16. In each stage, the first block has a stride of 2 for feature downsampling. Except for the first two stages, each stage contains up to 4 blocks for search. There are 14 layers in the backbone space, as listed in Tab. 1 (*i.e.*, the layers with a choice number of 6). This space contains about $6^{14} \approx 7.8 \times 10^{10}$ possible backbone architectures for search.

Head Space $\mathcal{A}_h$. A head architecture candidate contains two branches: one for classification while the other for regression. Both of them include at most 8 searchable layers (see Tab. 1). The first layer is a DSConv with kernel sizes of $\{3, 5\}$ and channel numbers of $\{128, 192, 256\}$. The subsequent 7 layers follow the same channel setting as the first layer, and have kernel choices of $\{3, 5\}$. An additional skip

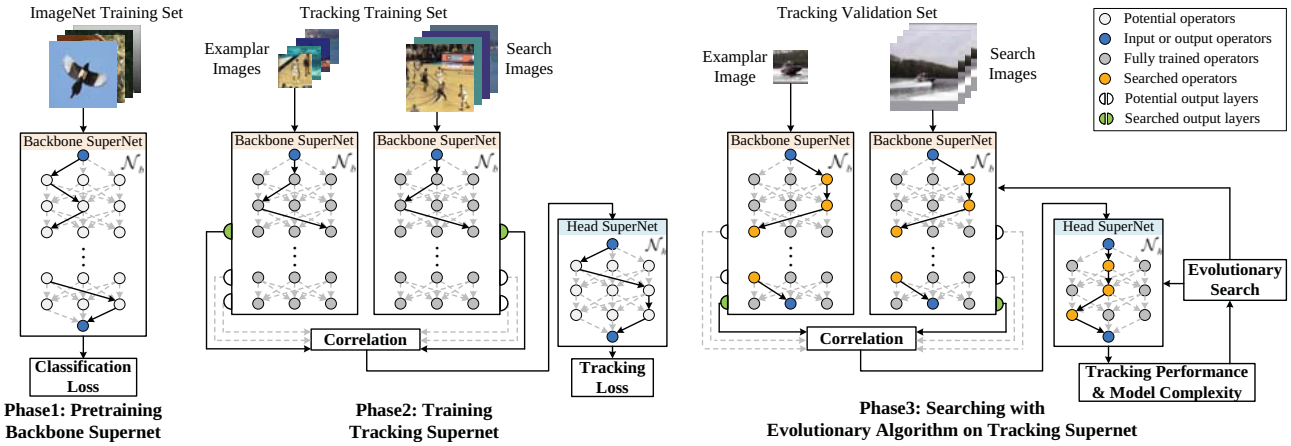

Figure 2: Search pipeline of the proposed LightTrack. There are three phases: pretraining backbone supernet, training tracking supernet, and searching with evolutionary algorithm on the tracking supernet. Better view in color with zoom-in.

connection is used to enable elastic depth of head architectures [58]. Different from the backbone space, the head does not include the kernel choice of 7 because the feature resolution has been relatively low. The head space contains about $(3 \times 3^8)^2 \approx 3.9 \times 10^8$ possible architectures for search.

In addition, at present, there is no definitive answer to the question of which layer's feature is more suitable for object tracking. We thereby add a new dimension in the search space to allow the one-shot method to determine the output feature layer automatically. Specifically, during supernet training, we randomly pick up an end layer from the last eight blocks in the backbone supernet, and use the output of the picked layer as the extracted feature. Such strategy is able to sample different possible blocks, and allows evolutionary search algorithm to evaluate which layer is better.

It is worth noting that the defined search space contains architectures ranging from 208M to 1.4G Flops with parameter sizes from 0.2M to 5.4M. Such space is much more lightweight than existing handcrafted networks. For example, the human-designed SiamRPN++ with ResNet-50 backbone has 48.9G FLOPs with 54M Params [22], being orders of magnitude more complex than architectures in the designed search space. This low-complexity space makes the proposed one-shot NAS algorithm easier to find promising lightweight architectures for tracking.

4.3. Search Pipeline

Our LightTrack includes three sequential phases: pretraining backbone supernet, training tracking supernet, and searching with evolutionary algorithm on the trained supernets. The overall pipeline is visualized in Fig. 2.

Phase 1: Pre-training Backbone Supernet. The backbone supernet $\mathcal{N}_b$ encodes all possible backbone networks in the search space $\mathcal{A}_b$. The structure of $\mathcal{N}_b$ is presented in Tab. 1. As defined in Eq. (2), the pre-training of the backbone supernet $\mathcal{N}_b$ is to optimize the cross-entropy loss on ImageNet. To decouple the weights of individual subnets, we perform uniform path sampling for the pre-training. In other words, in each batch, only one random path is sampled for feedforward and backward propagation, while other paths are frozen.

Phase 2: Training Tracking Supernet. The structure of the tracking supernet $\mathcal{N}$ is visualized in Fig. 2 (middle). In essence, it is a variant of Siamese tracker [30, 56]. Specifically, it takes a pair of tracking images as the input, comprising an exemplar image and a search image. The exemplar image represents the object of interest, while the search image represents the search area in subsequent video frames. Both inputs are processed by the pre-trained backbone network for feature extraction. The generated two feature maps are cross-correlated to generate correlation volumes. The head network contains one classification branch and one regression branch for object localization. The architecture of the head supernet can be found in Tab. 1.

The training also adopts the single-path uniform sampling scheme, but involving the tracking head and metrics. In each iteration, the optimizer updates one random path sampled from the backbone and head supernets. The loss function $\mathcal{L}_{train}^{trk}$ in Eq. (3) includes the common-used binary cross-entropy loss for foreground-background classification and the IoU loss [54] for object bounding-box regression.

Phase 3: Searching with Evolutionary Algorithm. The last phase is to perform evolutionary search on the trained supernet. Paths in the supernet are picked and evaluated under the direction of the evolutionary controller. At first, a population of architectures is initialized randomly. The top-k architectures are picked as parents to generate child networks. The next generation networks are generated by mutation and crossover. For crossover, two randomly selected candidates are crossed to produce a new

Table 2: Comparisons on VOT-19 [28]. (G) and (M) represent using GoogleNet and MobileNet-V2 as backbones, respectively. DiMP^r indicates the real-time version of DiMP, as reported in [28]. Ocean(off) denotes the offline version of Ocean [56]. Some values are missing because either the tracker is not open-resourced or the online update module does not support precise Flops estimation.

	SiamMask [50]	SiamFC++(G) [52]	SiamRPN++(M) [30]	ATOM [14]	TKU [48]	DiMP^r [6]	Ocean(off) [56]	**Ours** Mobile	**Ours** LargeA	**Ours** LargeB
EAO(↑)	0.287	0.288	0.292	0.301	0.314	0.321	0.327	**0.333**	**0.340**	**0.357**
Accuracy(↑)	**0.594**	0.583	0.580	**0.603**	0.589	0.582	**0.590**	0.536	0.540	0.552
Robustness(↓)	0.461	0.406	0.446	0.411	0.349	0.371	0.376	**0.321**	**0.315**	**0.310**
FLOPs(G)(↓)	15.5	17.5	7.0	-	-	-	20.3	**0.53**	**0.78**	**0.79**
Parameters(M)(↓)	16.6	13.9	11.2	8.4	-	26.1	25.9	**1.97**	**2.62**	**3.13**

one. For mutation, a randomly selected candidate mutates its every choice block with probability 0.1 to produce a new candidate. Crossover and mutation are repeated to generate enough new candidates that meet the given architecture constraints in Eq.(4).

One necessary detail is about Batch Normalization [26]. During search, subnets are sampled in a random way from the supernets. The issue is that the batch statistics on one path should be independent of others [20, 9]. Therefore, we need to recalculate batch statistics for each single path (subnet) before inference. We sample a random subset from the tracking training set to recompute the batch statistics for the single path to be evaluated. It is extremely fast and takes only a few seconds because no back-propagation is involved.

5. Experiments

5.1. Implementation Details

Search. Following the search pipeline, we first pre-train the backbone supernet on ImageNet for 120 epochs using the following settings: SGD optimizer with momentum 0.9 and weight decay 4e-5, initial learning rate 0.5 with linear annealing. Then, we train the head and the backbone supernets jointly on tracking data. The same as previous work [56], the tracking data consists of Youtube-BB [43], ImageNet VID [44], ImageNet DET [44], COCO [35] and the training split of GOT-10K [25]. The training takes 30 epochs, and each epoch uses 6×10^5 image pairs. The whole network is optimized using SGD optimizer with momentum 0.9 and weight decay 1e-4. Each GPU hosting 32 images, hence the mini-batch size is 256 images per iteration. The global learning rate increases linearly from 1e-2 to 3e-2 during the first 5 epochs and decreases logarithmically from 3e-2 to 1e-4 in the rest epochs. We freeze the parameters of the backbone in the first 10 epochs and set their learning rate to be $10\times$ smaller than the global learning rate in the rest epochs. Finally, to evaluate the performance of paths in the supernet, we choose the validation set of GOT-10K [25] as the evaluation data, since it does not have any overlap with both the training and the final test data.

Retrain. After evolutionary search, we first retrain the discovered backbone network for 500 epochs on Imagenet using similar settings as EfficientNet [46]: MSProp optimizer with momentum 0.9 and decay 0.9, weight decay 1e-5, dropout ratio 0.2, initial learning rate 0.064 with a warmup in the first 3 epochs and a cosine annealing, AutoAugment [12] policy and exponential moving average are adopted for training. Next, we fine-tune the discovered backbone and head networks on the tracking data. The fine-tuning settings in this step are similar to those of the supernet fine-tuning. The main differences include two aspects. 1) The searched architecture is trained for 50 epochs, which is longer than that of the tracking supernet fine-tuning. (2) The global learning rate increases from 2e-2 to 1e-1 during the first 5 epochs and then decreases from 1e-1 to 2e-4 in the rest epochs.

Test. The inference follows the same protocols as in [5, 31]. The feature of the target object is computed once at the first frame, and then consecutively matched with subsequent search images. The hyper-parameters in testing are selected with the tracking toolkit [56], which contains an automated parameter tuning algorithm. Our trackers are implemented using Python 3.7 and PyTorch 1.1.0. The experiments are conducted on a server with 8 Tesla V100 GPUs and a Xeon E5-2690 2.60GHz CPU.

5.2. Results and Comparisons

We compare LightTrack to existing hand-designed object trackers with respect to model performance, complexity and run-time speed. The performance is evaluated on four benchmarks, including VOT-19 [28], GOT-10K [25], TrackingNet [39] and LaSOT [16], while the speed is tested on resource-constrained hardware platforms, involving Apple iPhone7 PLUS, Huawei Nova 7 5G, and Xiaomi Mi 8. Moreover, we provide three versions of LightTrack under different resource constraints, *i.e.*, LightTrack Mobile ($\leq$600M Flops, $\leq$2M Params), LargeA ($\leq$800M Flops, $\leq$3M Params) and LargeB ($\leq$800M Flops, $\leq$4M Params).

VOT-19. This benchmark contains 60 challenging sequences, and measures tracking accuracy and robustness simultaneously by expected average overlap (EAO). As reported in Tab. 2, LightTrack-Mobile achieves superior performance compared to existing SOTA offline trackers, such as SiamRPN++ [30] and SiamFC++ [52], while using $>$10 times fewer model Flops and Params. Furthermore, com-

Table 3: Comparisons on GOT-10k [25]. (R) and (G) represents ResNet-50 and GoogleNet, respectively.

	DaSiam [57]	SiamRPN++(R) [30]	ATOM [14]	Ocean-offline [56]	SiamFC++(G) [52]	Ocean-online [56]	DiMP-50 [6]	**Ours** Mobile	**Ours** LargeA	**Ours** LargeB
AO(↑)	0.417	0.518	0.556	0.592	0.595	**0.611**	**0.611**	**0.611**	**0.615**	**0.623**
SR0.5(↑)	0.461	0.618	0.634	0.695	0.695	**0.721**	0.712	0.710	**0.723**	**0.726**
FLOPs(G)(↓)	21.0	48.9	-	20.3	17.5	-	-	**0.53**	**0.78**	**0.79**
Parameters(M)(↓)	19.6	54.0	8.4	25.9	13.9	44.3	26.1	**1.97**	**2.62**	**3.13**

Table 4: Comparisons on TrackingNet *test* set [39]. (A) and (R) represent AlexNet and ResNet-50, respectively.

	RTMDNet [27]	ECO [13]	DaSiam [57]	C-RPN [17]	ATOM [14]	SiamFC++(A) [52]	SiamRPN++(R) [30]	DiMP-50 [6]	**Ours** Mobile	**Ours** LargeA	**Ours** LargeB
P(%)	53.3	55.9	59.1	61.9	64.8	64.6	69.4	68.7	**69.5**	**70.0**	**70.8**
P_{norm}(%)	69.4	71.0	73.3	74.6	77.1	75.8	**80.0**	**80.1**	77.9	78.8	**78.9**
AUC(%)	58.4	61.2	63.8	66.9	70.3	71.2	**73.3**	**74.0**	72.5	**73.6**	**73.3**

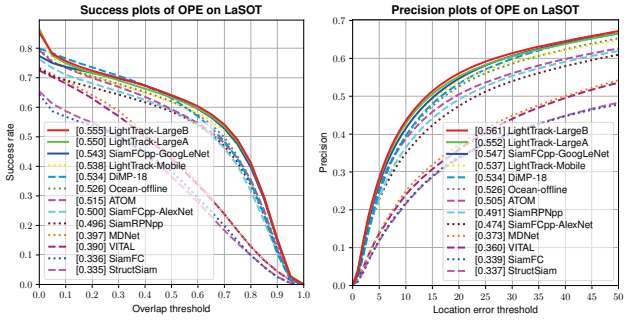

Figure 3: Comparisons on LaSOT *test* dataset [16].

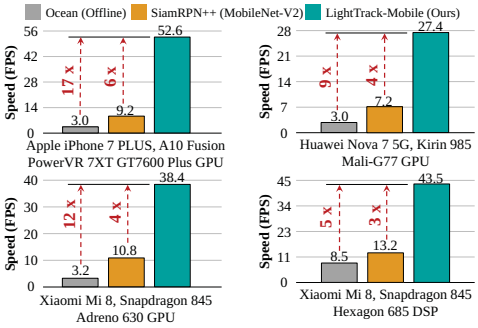

Figure 4: Run-time speed on resource-limited platforms.

pared to the trackers with online update, such as ATOM [14] and DiMP^r [6], LightTrack-LargeB is also competitive, surpassing them by 5.6% and 3.6% respectively. This demonstrates the efficacy of the proposed one-shot search algorithm and the discovered architecture.

GOT-10K. GOT-10K [25] is a new benchmark covering a wide range of common challenges in object tracking, such as deformation and occlusion. Tab. 3 shows that LightTrack obtains state-of-the-art performance, compared to current prevailing trackers. The AO score of LightTrack-Mobile is 1.6% and 1.9% superior than SiamFC++(G) [52] and Ocean(off) [56], respectively. Besides, if we loosen the computation constraint, the performance of LightTrack will be further improved. For example, LightTrack-LargeB outperforms DiMP-50 [6] by 1.2%, while using 8× fewer Params (3.1 *v.s.* 26.1 M).

TrackingNet. TrackingNet [39] is a large-scale short-term tracking benchmark containing 511 video sequences in *test* set. Tab. 4 presents that LightTrack-Mobile achieves better precision (69.5%), being 0.8% higher than DiMP-50 [6]. Besides, the P_{norm} and AUC of LightTrack-Mobile are comparable to SiamRPN++ and DiMP-50, while using 96% and 92% fewer model Params, respectively.

LaSOT. LaSOT [16] is by far the largest single object tracking benchmark with high-quality frame-level annotations. As shown in Fig. 3, LightTrack-LargeB achieves a success score of 0.555, which surpasses SiamFC++(G) [52] and Ocean-offline [56] by 1.2% and 2.9%, respectively. Compared to the online DiMP-18 [6], LightTrack-LargeB improves the success score by 2.1%, while using 12× fewer Params (3.1 *v.s.* 39.3 M).

Speed. Fig. 4 summarizes the run-time speed of LightTrack on resource-limited mobile platforms, , including Apple iPhone 7 Plus, Huawei Nova 7 and Xiaomi Mi 8. We observe that SiamRPN++ [30] and Ocean [56] cannot run at real-time speed (*i.e.*, $<$ 25 *fps*) on these edge devices, such as Snapdragon 845 Adreno 630 GPU and Hexagon 685 DSP. In contrast, our LightTrack run much more efficiently, being **3∼6× faster** than SiamRPN++ (MobileNetV2 backbone), and **5∼17× faster** than Ocean (offline) on Snapdragon 845 GPU and DSP [3], Apple A10 Fusion PowerVR GPU [1], and Kirin 985 Mali-G77 GPU [2]. The real-time speed allows LightTrack to be deployed and applied in resource-constrained applications, such as camera drones where edge chipsets are commonly used. The speed improvements also demonstrate that LightTrack is effective and can find more compact and efficient object trackers.

5.3. Ablation and Analysis

Component-wise Analysis. We evaluate the effects of different components in our LightTrack on VOT-19 [28], and report the results in Tab. 5. Our baseline is a hand-crafted mobile tracker, which takes MobileNetV3-large [23] as the backbone (chopping off the last stage), and outputs features from the last layer with a stride of 16. The head network stacks 8 layers of depthwise separable con-

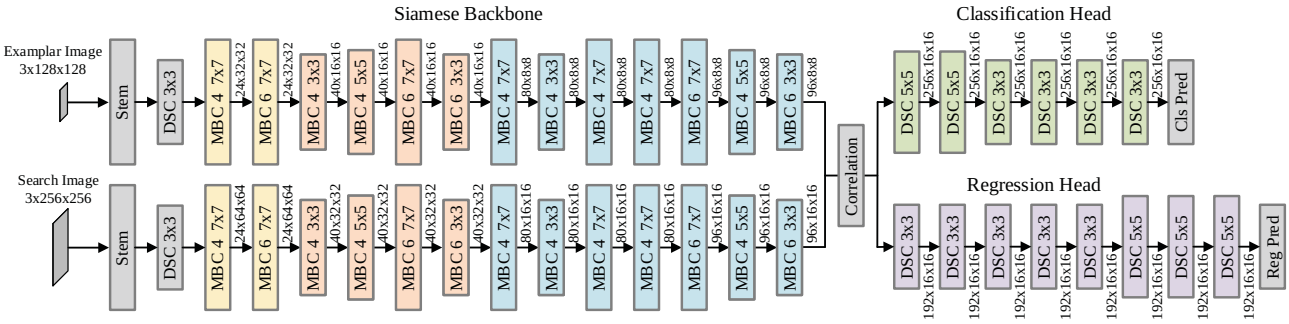

Figure 5: The architecture searched by the proposed LightTrack (Mobile). The searchable layers are drawn in colors while the fixed/pre-defined parts are plotted in grey. The "Stem" consists of a normal 2D convolution layer with kernel size of 3×3 and stride of 2, a BatchNorm layer, and a Swish activation layer. "DSConv" indicates depthwise separable convolution [11] while "MBConv" denotes mobile inverted bottleneck [45] with squeeze excitation [24].

volution (DSConv) [11] in both classification and regression branches. For each DSConv, the kernel size is set to 3×3 and the number of channels is 256. The EAO performance of the baseline is 0.268. For ablation, we add the components in the baseline into search and change the handcrafted architectures with automatically searched ones. As presented in Tab. 5 #2, when the backbone architecture is automatically searched, the EAO performance is improved by 2.4%. This demonstrates that the hand-designed MobileNetV3-large backbone is not optimal for object tracking, because it is primarily designed for image classification, where the precise localization of the object is not paramount. If we add the output feature layer into search, the performance is further improved to 0.307. This shows that our method can search out a better layer for feature extraction. The comparison between #4 and #1 shows that the searchable head architecture is superior to the handcrafted one, inducing 2.9% EAO gains. When searching the three components together, as shown in #5, the complete LightTrack achieves better performance than only searching parts of the tracking network.

Impact of ImageNet Pre-training. We pre-train the searched architecture on ImageNet for 0, 200 and 500 epochs, and evaluate their impact for final tracking performance. As reported in Tab. 6, no pre-training has a significantly negative impact on tracking accuracy. Better pre-training allows the tracker to achieve higher performance.

Analysis of Searched Architecture. Fig. 5 visualizes the LightTrack-Mobile architecture searched by the proposed one-shot NAS method. We observe several interesting phenomena. 1) There are about 50% of the backbone blocks using MBConv with kernel size of 7x7. The underlying reason may be that large receptive fields can improve the localization precision. 2) The searched architecture chooses the second-last block as the feature output layer. This may reveals that tracking networks might not prefer high-level features. 3) The classification branch contains fewer layers than the regression branch. This may be attributed to the fact that coarse object localization is relatively easier than precise bounding box regression. These findings might enlighten future works on designing new tracking networks.

Table 5: Ablation for searchable components. ✓ indicates automatically searched, while ✗ denotes hand-designed.

#	Backbone	Output Layer	Head	EAO
1	✗	✗	✗	0.268
2	✓	✗	✗	0.292
3	✓	✓	✗	0.307
4	✗	✗	✓	0.297
5	✓	✓	✓	0.333

Table 6: Impact of ImageNet Pre-training.

	Epoch 0	Epoch 200	Epoch 500
Top-1 Acc (%)	–	72.4	77.6
EAO on VOT-19 (%)	21.3	31.2	33.3

6. Conclusion

This paper makes the first effort on designing lightweight object trackers via neural architecture search. The proposed method, *i.e.*, LightTrack, reformulates one-shot NAS specialized for object tracking, as well as introducing an effective search space. Extensive experiments on multiple benchmarks show that LightTrack achieves state-of-the-art performance, while using much fewer Flops and parameters. Besides, LightTrack can run in real-time on diverse resource-restricted platforms. We expect this work might be able to narrow the gap between academic methods and industrial applications in object tracking field.

Acknowledgement. We would like to thank the reviewers for their insightful comments. Lu and Wang are supported in part by the National Key R&D Program of China under Grant No. 2018AAA0102001 and National Natural Science Foundation of China under grant No. 61725202, U1903215, 61829102, 91538201, 61771088, 61751212 and Dalian Innovation leader's support Plan under Grant No. 2018RD07.

References

[1] https://en.wikipedia.org/wiki/Apple_A10. 7

[2] https://www.hisilicon.com/en/products/Kirin/Kirin%20985. 7

[3] https://www.qualcomm.com/products/snapdragon-845-mobile-platform. 2, 7

[4] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *ICML*, 2018. 2

[5] Luca Bertinetto, Jack Valmadre, João F Henriques, Andrea Vedaldi, and Philip H S Torr. Fully-convolutional siamese networks for object tracking. In *ECCVW*, 2016. 1, 2, 6

[6] Goutam Bhat, Martin Danelljan, Luc Van Gool, and Radu Timofte. Learning discriminative model prediction for tracking. In *ICCV*, 2019. 2, 3, 6, 7

[7] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *ICLR*, 2019. 2

[8] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *ICCV*, 2019. 2

[9] Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Xinyu Xiao, and Jian Sun. DetNAS: Backbone search for object detection. In *NIPS*, 2019. 2, 4, 6

[10] Zedu Chen, Bineng Zhong, Guorong Li, Shengping Zhang, and Rongrong Ji. Siamese box adaptive network for visual tracking. In *CVPR*, 2020. 2

[11] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, 2017. 2, 3, 4, 8

[12] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *CVPR*, 2019. 6

[13] Martin Danelljan, Goutam Bhat, Fahad Shahbaz Khan, and Michael Felsberg. ECO: Efficient convolution operators for tracking. In *CVPR*, 2017. 7

[14] Martin Danelljan, Goutam Bhat, Fahad Shahbaz Khan, and Michael Felsberg. ATOM: Accurate tracking by overlap maximization. In *CVPR*, 2019. 2, 3, 6, 7

[15] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *JMLR*, 20(55):1–21, 2019. 1, 2

[16] Heng Fan, Liting Lin, Fan Yang, Peng Chu, Ge Deng, Sijia Yu, Hexin Bai, Yong Xu, Chunyuan Liao, and Haibin Ling. LaSOT: A high-quality benchmark for large-scale single object tracking. In *CVPR*, 2019. 6, 7

[17] Heng Fan and Haibin Ling. Siamese cascaded region proposal networks for real-time visual tracking. In *CVPR*, 2019. 7

[18] Junyu Gao, Tianzhu Zhang, and Changsheng Xu. Graph convolutional tracking. In *CVPR*, 2019. 2

[19] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. NAS-FPN: Learning scalable feature pyramid architecture for object detection. In *CVPR*, 2019. 2

[20] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *ECCV*, 2020. 2, 3, 4, 6

[21] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*, 2016. 1

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 2, 5

[23] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *ICCV*, 2019. 2, 3, 4, 7

[24] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, 2018. 4, 8

[25] Lianghua Huang, Xin Zhao, and Kaiqi Huang. GOT-10k: A large high-diversity benchmark for generic object tracking in the wild. *TPAMI*, 2019. 6, 7

[26] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 6

[27] Ilchae Jung, Jeany Son, Mooyeol Baek, and Bohyung Han. Real-Time MDNet. In *ECCV*, 2018. 7

[28] Matej Kristan, Jiri Matas, Ales Leonardis, et al. The seventh visual object tracking VOT2019 challenge results. In *ICCVW*, 2019. 6, 7

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 2

[30] Bo Li, Wei Wu, Qiang Wang, Fangyi Zhang, Junliang Xing, and Junjie Yan. SiamRPN++: Evolution of siamese visual tracking with very deep networks. In *CVPR*, 2019. 1, 2, 5, 6, 7

[31] Bo Li, Junjie Yan, Wei Wu, Zheng Zhu, and Xiaolin Hu. High performance visual tracking with siamese region proposal network. In *CVPR*, 2018. 1, 2, 3, 6

[32] Feng Li, Cheng Tian, Wangmeng Zuo, Lei Zhang, and Ming-Hsuan Yang. Learning spatial-temporal regularized correlation filters for visual tracking. In *CVPR*, 2018. 2

[33] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *UAI*, 2019. 2, 3

[34] Xin Li, Chao Ma, Baoyuan Wu, Zhenyu He, and Ming-Hsuan Yang. Target-aware deep tracking. In *CVPR*, 2019. 2

[35] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, 2014. 6

[36] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Auto-Deeplab: Hierarchical neural architecture search for semantic image segmentation. In *CVPR*, 2019. 2

[37] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture searc. In *ICLR*, 2019. 2

[38] Yuanpei Liu, Xingping Dong, Wenguan Wang, and Jianbing Shen. Teacher-students knowledge distillation for siamese trackers. *arXiv preprint arXiv:1907.10586*, 2019. 1

[39] Matthias Muller, Adel Bibi, Silvio Giancola, Salman Alsubaihi, and Bernard Ghanem. Trackingnet: A large-scale

dataset and benchmark for object tracking in the wild. In *ECCV*, 2018. 6, 7

[40] Hyeonseob Nam and Bohyung Han. Learning multi–domain convolutional neural networks for visual tracking. In *CVPR*, 2016. 2

[41] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *ICML*, 2018. 2

[42] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI*, 2019. 3, 4

[43] Esteban Real, Jonathon Shlens, Stefano Mazzocchi, Xin Pan, and Vincent Vanhoucke. Youtube-boundingboxes: A large high-precision human-annotated data set for object detection in video. In *CVPR*, 2017. 6

[44] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, and Michael Bernstein. ImageNet Large scale visual recognition challenge. *IJCV*, 2015. 6

[45] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018. 2, 3, 4, 8

[46] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019. 6

[47] Ran Tao, Efstratios Gavves, and Arnold W. M. Smeulders. Siamese instance search for tracking. In *CVPR*, 2016. 2

[48] Ardhendu Shekhar Tripathi, Martin Danelljan, Luc Van Gool, and Radu Timofte. Tracking the known and the unknown by leveraging semantic information. In *BMVC*, 2019. 6

[49] Paul Voigtlaender, Jonathon Luiten, Philip HS Torr, and Bastian Leibe. Siam R-CNN: Visual tracking by re-detection. In *CVPR*, 2020. 2

[50] Qiang Wang, Li Zhang, Luca Bertinetto, Weiming Hu, and Philip H. S. Torr. Fast online object tracking and segmentation: A unifying approach. In *CVPR*, 2019. 6

[51] Lingxi Xie and Alan Yuille. Genetic cnn. In *ICCV*, 2017. 2

[52] Yinda Xu, Zeyu Wang, Zuoxin Li, Ye Yuan, and Gang Yu. SiamFC++: towards robust and accurate visual tracking with target estimation guidelines. In *AAAI*, 2020. 1, 2, 6, 7

[53] Tianyu Yang, Pengfei Xu, Runbo Hu, Hua Chai, and Antoni B Chan. ROAM: Recurrently optimizing tracking model. In *CVPR*, 2020. 2

[54] Jiahui Yu, Yuning Jiang, Zhangyang Wang, Zhimin Cao, and Thomas Huang. Unitbox: An advanced object detection network. In *ACM MM*, 2016. 5

[55] Zhipeng Zhang and Houwen Peng. Deeper and wider siamese networks for real-time visual tracking. In *CVPR*, 2019. 1

[56] Zhipeng Zhang, Houwen Peng, Jianlong Fu, Bing Li, and Weiming Hu. Ocean: Object-aware anchor-free tracking. In *ECCV*, 2020. 1, 2, 5, 6, 7

[57] Zheng Zhu, Qiang Wang, Bo Li, Wei Wu, Junjie Yan, and Weiming Hu. Distractor-aware siamese networks for visual object tracking. In *ECCV*, 2018. 7

[58] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *ICLR*, 2017. 2, 5