NAS for Object Detection

本文大致介绍了四种将 NAS 应用于目标检测任务上的论文。为之后的工作提供思路与参考。

SM-NAS

原文：
https://ojs.aaai.org/index.php/AAAI/article/view/6958/6812

这篇 Paper 要做的事情是将神经网络结构搜索和目标检测相结合，而且是从目标检测算法的整个 pipeline 角度来搜。作者用了 coarse-to-fine 的思想，把 NAS 的搜索过程分为两个 stage，先进行结构层次 (structural-level) 的搜索，再进行模块级别 (modular-level) 的搜索，下面分别进行介绍：

结构搜索（Structural-level）

搜索空间如下：

backbone :ResNet 系列 (Res18, 34, 50, 101), ResNeXt 系列 (50, 101), MobileNet V2 。并且都用了 ImageNet 初始化

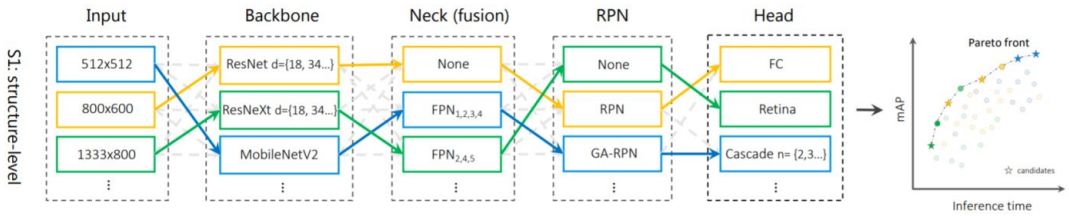
neck: no FPN （也就是 C4），FPN 从 P1 到 P6 的哪层进入哪层出 (P1-P5，P2-P5 之类的)

RPN: no RPN (one-stage)，RPN，Guided Anchor

head: RCNN head, RetinaNet head, Cascade RCNN head (包括 cascade 的数量从 2 ~ 4)

输入图片分辨率：[512x512，800x600，1080x720，1333x800]

这部分评价指标是 inference-time 和准确率。



作者最后得到了 11000 种候选的组合，用进化算法来搜，先随机生成初始值，然后选种变异之类的。最后花了 2000 GPU hours，一共搜了约 500 种组合。得到的结果如下图所示：

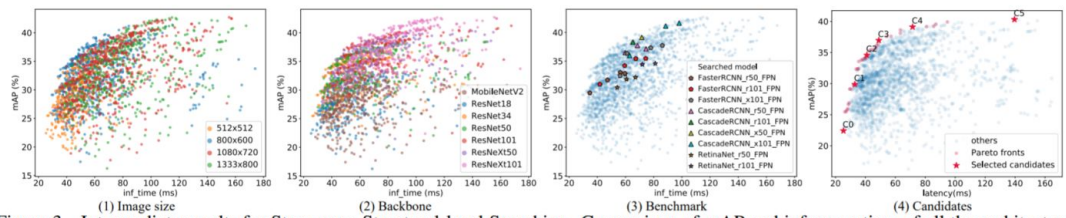


Figure 3. Intermediate results for Stage-one: Structural-level Searching. Comparison of mAP and inference time of all the architectures searched on COCO. Inference time is tested on one V100 GPU. It can be found our searching already found many structures dominate state-of-the-art objectors. On the Pareto front, we pick 6 models (C0 to C5) and further search for better modular-level architectures in Stage-two.

这里在看别人的论文解读的时候，有人提出来一个观点，我觉得挺有道理的所以搬过来

了：

这里有一个和其他论文不太一样的结论，在第 2 幅小图中，MobileNet v2 是不在帕累托前沿面上的，论文解释是说 depthwise 卷积在 GPU 上优化不好导致的，虽然计算量低，但是效率不高，该论文采用 inference time 作为评价指标。但是这里其实存在一个不公平的地方，所有的主干网络模型是用 ImageNet 预训练模型的，而 MobileNet v2 的预训练模型的 weight decay 是 4e-5，和 ResNet 的 1e-4 是不一样的，而在整个搜索中是采用 1e-4 的，这对 MobileNet v2 的性能是有影响的（可能会有 1 个点），也就是上说预训练 MobileNet v2 和搜索采用的 weight decay 是不适配的，这个结论对比不够公平。

模型搜索（Modular-level）

在上一步确定了选择哪些模块、模块之间的连接方式之后，这一步来单独优化每个模块。

作者先搜一个 backbone ，固定了有 5 个 stage，并且固定了要逐渐 downsample 2 倍。由于换新的 backbone 再用 ImageNet 初始化会很麻烦，作者在这里就都不用 pre-train model，随机初始化了。为了减轻不初始化的影响，作者把 Batch Normalization 替换成 Group Normalization ，并加入了 Weight Standardization。

这一部分的参数空间非常简单，其实就是搜 backbone 的 channel 数量。比如在之前的 structural-level 搜到的 C5 模型，backbone 用的是 X101_bottleneck。作者把初始的 channel 数量指定为 {48, 56, 64, 72} 四个候选值，然后搜后面的 conv 里面要在哪里把 channel 数量乘 2。并没有引入新的运算。作者还搜了 FPN 和 head 部分的 channel 数量，候选组合就是 {128, 256, 512} 。这一步作者又把衡量速度的指标从 inference time 改成了 FLOPs，作者说 inference time 在不同 GPU 上不一样，而 FLOPs 相对更 consistency。

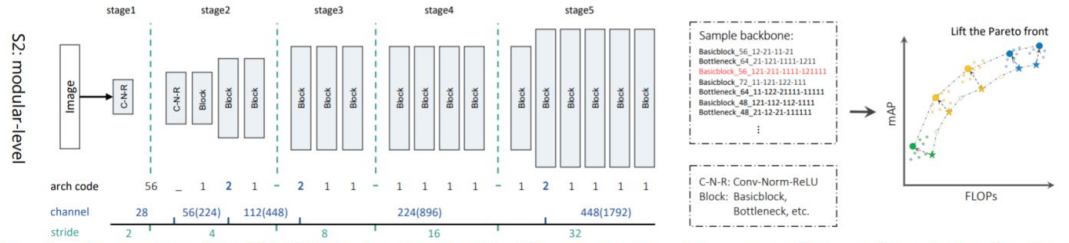


Figure 2. An overview of our SM-NAS for detection pipeline. We propose a two-stage coarse-to-fine searching strategy directly on detection dataset: S1: Structural-level searching stage first aims to finding an efficient combination of different modules; S2: Modular-level search stage then evolves each specific module and push forward to a faster task-specific network.

最后搜出来的结果，E0 - E5 如下表所示：

Model	Input size	Backbone	Neck	RPN	RCNN Head	Backbone FLOPs	Time (ms)	mAP
E0	512x512	basicblock_64_1-21-21-12	FPN(P_2-P_5, c=128)	RPN	2FC	7.2G (0.75)	24.5	27.1
E1	800x600	basicblock_56_111-2111-2-111112	FPN(P_2-P_5, c=256)	RPN	2FC	28.3G (0.79)	32.2	34.3
E2	800x600	basicblock_48_12-11111-211-1112	FPN(P_1-P_5, c=128)	RPN	Cascade(n=3)	23.8G (0.67)	39.5	40.1
E3	800x600	bottleneck_56_211-111111111-2111111-11112111	FPN(P_1-P_5, c=128)	RPN	Cascade(n=3)	59.2G (0.78)	50.7	42.7
E4	800x600	Xbottleneck_56_21-21-1111111111111111-2111111	FPN(P_1-P_5, c=256)	GA-RPN	Cascade(n=3)	73.5G (0.96)	80.2	43.9
E5	1333x800	Xbottleneck_56_21-21-1111111111111111-21111111	FPN(P_1-P_5, c=256)	GA-RPN	Cascade(n=3)	162.45G (0.94)	108.1	46.1

对每个子结构进行了搜索，其性能都有较大的提升。可以看到 two-stage 的方法（特别是 cascade head）占据了较大的优势地位。

最后结果如下：

Method	Backbone	Input size	Inf time (ms)	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
YOLO v3[44]	DarkNet-53	608x608	51.0 (TitanX)	33.0	57.9	34.4	18.3	35.4	41.9
DSSD513[13]	ResNet101	513x513	-	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet[28]	ResNet101-FPN	1333x800	91.7 (V100)	39.1	59.1	42.3	21.7	42.7	50.2
FSAF[62]	ResNet101-FPN	1333x800	92.5 (V100)	40.9	61.5	44.0	24.0	44.2	51.3
CornerNet[19]	Hourglass-104	512x512	244.0 (TitanX)	40.5	56.5	43.1	19.4	42.7	53.9
CenterNet[61]	Hourglass-104	512x512	126.0 (V100)	42.1	61.1	45.9	24.1	45.5	52.8
AlignDet[9]	ResNet101-FPN	1333x800	110.0 (P100)	42.0	62.4	46.5	24.6	44.8	53.3
GA-Faster RCNN[52]	ResNet50-FPN	1333x800	104.2 (V100)	39.8	59.2	43.5	21.8	42.6	50.7
Faster-RCNN[45]	ResNet101-FPN	1333x800	84.0 (V100)	39.4	-	-	-	-	-
Mask-RCNN[16]	ResNet101-FPN	1333x800	105.0 (V100)	40.2	-	-	-	-	-
Cascade-RCNN[5]	ResNet101-FPN	1333x800	97.9 (V100)	42.8	62.1	46.3	23.7	45.5	55.2
TridentNet[22]	ResNet101	1333x800	588 (V100)	42.7	63.6	46.5	23.9	46.4	55.6
TridentNet[22]	ResNet101-deformable-FPN	1333x800	2498.3 (V100)	48.4	69.7	53.5	31.8	51.3	60.3
DetNAS[10]	Searched Backbone	1333x800	-	42.0	63.9	45.8	24.9	45.1	56.8
NAS-FPN[14]	ResNet50-FPN(@384)	1280x1280	198.7 (V100)	45.4	-	-	-	-	-
SM-NAS: E2	Searched Backbone	800x600	**39.5**(V100)	**40.0**	58.2	43.4	21.1	42.4	51.7
SM-NAS: E3	Searched Backbone	800x600	**50.7**(V100)	**42.8**	61.2	46.5	23.5	45.5	55.6
SM-NAS: E5	Searched Backbone	1333x800	**108.1**(V100)	**45.9**	64.6	49.6	27.1	49.0	58.0

总结：

这篇 paper 用 NAS 的方法来搜整个目标检测的 pipeline。优点在于分两个 stage 来搜，coarse to fine 的思想，对于目标检测这种 pipleline 特别长的任务可以减小搜索空间。

Auto-FPN

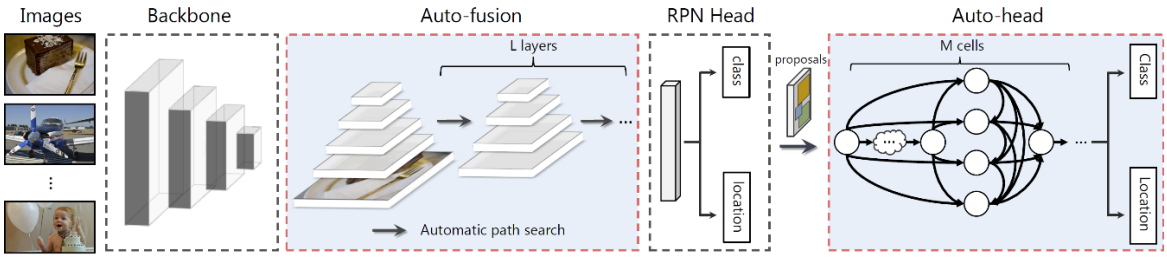
原文：

http://openaccess.thecvf.com/content_ICCV_2019/papers/Xu_Auto-FPN_Automatic_Network_Architecture_Adaptation_for_Object_Detection_Beyond_Classification_ICCV_2019_paper.pdf

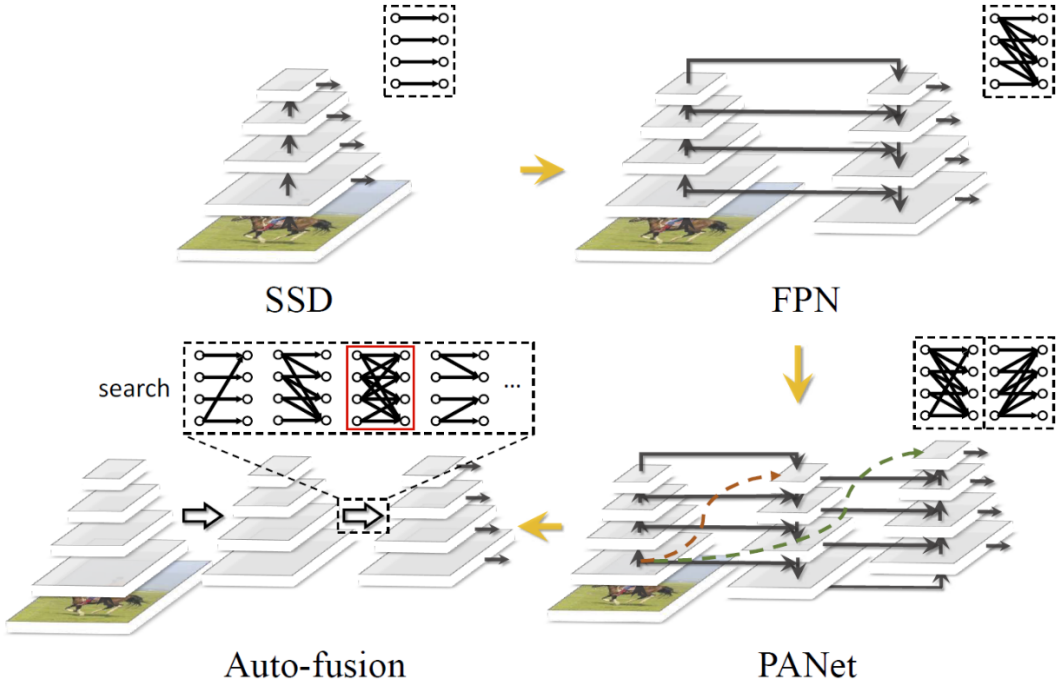
首先大致介绍一下 two-stage 目标检测网络的结构组成。

- **backbone**：即用来提取图像特征的网络结构，常用 ResNet 或 VGG 来提取特征
- **RPN(region proposal networks)**: 这个网络的作用是在 backbone 提取得到的特征的基础上进一步预测出若干个区域，还需要进一步由后面的网络处理
- **RCNN (region CNN) head**:针对生成的 proposal 区域做进一步的分类和位置回归。

Auto-FPN 的创新点在后面两个网络中，主要是针对 backbone 的特征先做自动 fusion 操作（**Auto-fusion**），另外就是对 head 网络使用 NAS 技术搜索得到一个网络用于分类和回归（**Auto-head**）。



Auto-fusion 其实是基于之前的一些工作的改进，如下图示：



- 最开始 SSD 会对不同尺度的特征图做预测
- 之后 FPN 提出了自上而下的连接方式，将不同尺度的特征图信息进行结合
- PANet 通过直接将最底层的特征与最高层的特征图进行连接，进一步增强特征图之间的联系
- Auto-fusion 则是通过搜索网络结构来让算法自动找到不同层之间最合适的连接方式。

Auto-fusion 搜索方法设计考虑的有如下两个方面：

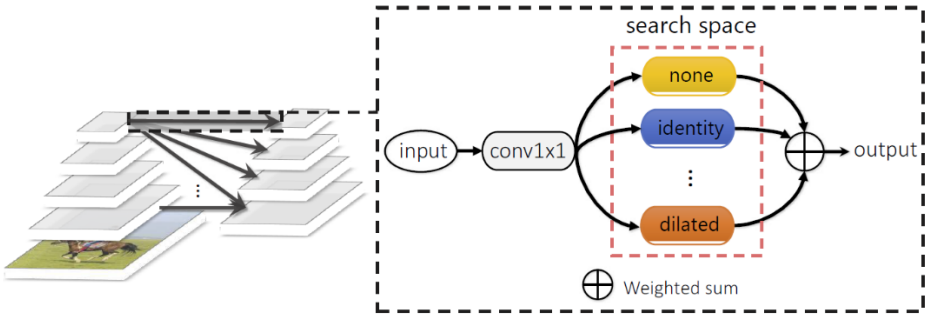
- 搜索空间覆盖所有的连接方式
- 因为在 TridenNet 中使用 **dilated conv** 得到了不错的效果，所以不同大小的 **dilated conv** 加入到了搜索空间中去。搜索空间如下：
  - **no connection (none)**
  - **skip connection (identity)**
  - **3×3 dilated conv with rate 2**
  - **5×5 dilated conv with rate 2**
  - **3×3 dilated conv with rate 3**
  - **5×5 dilated conv with rate 3**
  - **3×3 depthwise-separable conv**
  - **5×5 depthwise-separable conv**

对于 backbone 提取的 4 个特征图我们用 P_1^0、P_2^0、P_3^0、P_4^0表示，P_4^0 表示第 0 层的第 4 个特征图，其高宽都比原图的小 4 倍。那么如果 Auto-fusion 结构一共有 ℓ层，则第 ℓ层的特征图可表示为 P_1^l、P_2^l、P_3^l、P_4^l。第 ℓ-1 层的 i 节点到第 ℓ层的 j 节点的 operation 可表示为

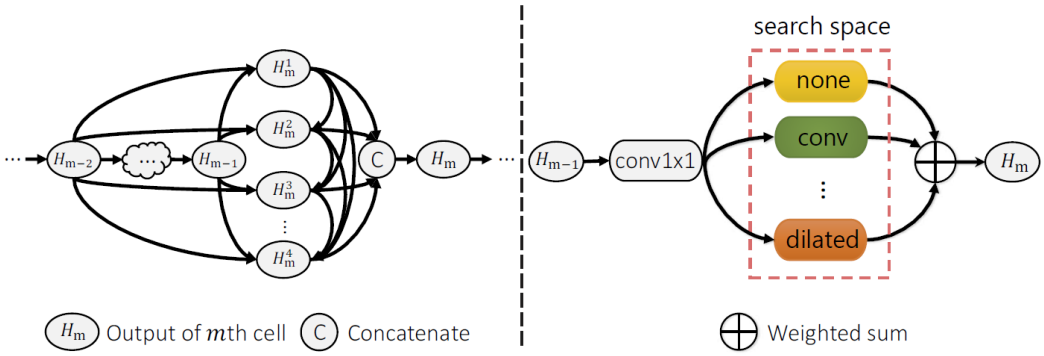
$$\hat{O}_{i\to j}(P_j^{l-1}) = \sum_{O^k \in \mathcal{O}_\mathcal{N}} \alpha_{i\to j}^{kl} O_{i\to j}^{kl}(P_j^{l-1})$$

$$\text{where } \textstyle\sum_{k=1}^{|\mathcal{O}_\mathcal{N}|} \alpha_{i\to j}^{kl} = 1, \text{ and } \alpha_{i\to j}^{kl} \geq 0.$$

下图给出了 Auto-fusion 某一层的示意图，可以看到因为要使得不同尺度的特征图能做融合操作，首先都会用 **conv 1*1** 的操作，而后再计算不同 operation 的权重。



Auto-head 就是用 NAS 搜索得到一个 CNN 结构，示意图如下



不过有如下几个方面作了修改：

- Auto-head 由 m 个 cell 组成，每个 cell 由 7 个 nodes 组成。每个 cell 有两个 inputs，分别是前一个和前前一个 cell 的 outputs。
- 在 DARTS 中，最后有两类 cell，分别是 normal cell 和 reduction cell。而在 **Auto-head 中每个 cell 的结构可以不一样**，也就是说 Auto-head 可以由 m 个不同的 cell 组成
- **Auto-head 中没有使用 reduction cell**，因为到了这一步特征图大小已经很小了（如 7*7）
- 同上，**涉及到卷积的操作也不用 dilated conv 了**，但是为了更好地提取特征，在**每个 operation 后面还会加上 3*3 和 5*5 的卷积操作**。

AutoFPN 还加入了**资源约束**，这样既可以避免生成的模型过大，也能够加速搜索和训练过程。很直观的一种想法是将 forward 时间作为约束条件，但是测得的 forward 时间相对于模型参数并不是可微的，所以考虑如下三个方面来对资源约束建模：

- 模型大小

- FLOPs
- memory access cost (MAC)

公式表示如下：

$$C(\alpha,\beta)=\sum_{i,j,k,l}\alpha_{i\to j}^{kl}C(O_{i\to j}^{kl})+\sum_{i,j,k}\beta_{i\to i}^{k}C(O_{i\to j}^{k}),$$

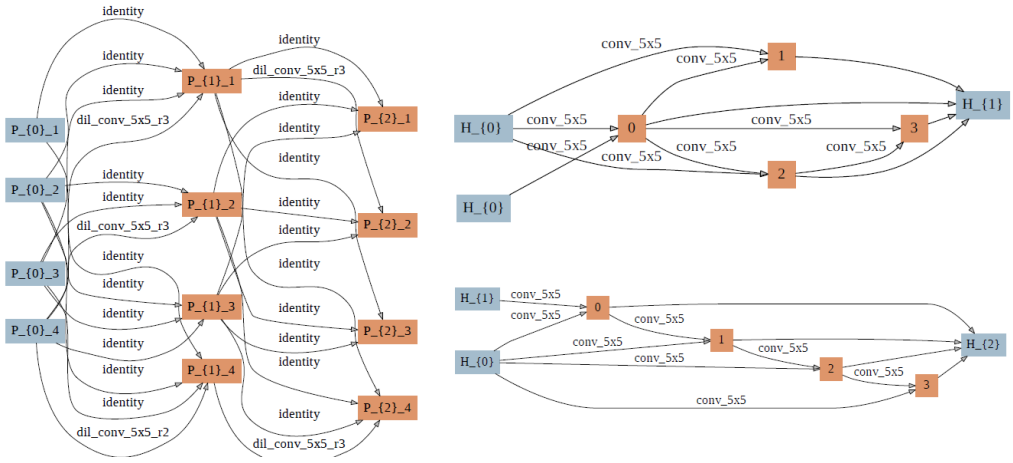
那么总的 loss 表达式如下：

$$\mathcal{L}(w,\alpha,\beta)=\mathcal{L}_{model}(w,\alpha,\beta)+\lambda C(\alpha,\beta)$$

其中α表示**模型参数，即我们常说的卷积核参数或者全连接层参数等**，β表示模型结构参数，即记录了不同 operation 的权重。通过修改 λ 这个参数的大小，我们能够控制模型的相对大小，例如如果我们令 λ 比较大，那么最终得到的网络大小就会相对小一些，反之则大一些。

Auto-fusion 和 Auto-head 搜索结果

下图给出了 Auto-fusion 和 Auto-head 搜索结果，可以看到 identity 和 conv_5*5 在两个结构中用的最多。



不同大小模型的结果对比

按照文中的说法是分别 Auto-fusion 和 Auto-head 是分别进行搜索的，所以最后的 AutoFPN 是将二者最好的结果进行组合得到的。可以看到在不同的数据集上搜索得到的网络结构更小，而且结果也能更好。

method	Search On	Params (neck)/M	Params (head)/M	Params (total)/M	mAP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
FPN	-	3.34	14.31	41.76	38.6	60.4	42.0	23.7	42.4	49.2
Auto-fusion$_S$	COCO	1.39	14.31	39.8	39.3	61.7	42.5	24.9	43.5	50
Auto-fusion$_M$	COCO	1.61	14.31	40.02	39.6	61.9	42.8	24.6	44.0	50.3
Auto-fusion$_L$	COCO	1.83	14.31	40.24	39.7	62.0	42.9	25.7	43.8	50.4
Auto-head$_S$	COCO	3.34	2.42	29.87	39.1	59.9	42.6	23.2	43.3	49.1
Auto-head$_M$	COCO	3.34	6.93	34.37	39.9	60.5	43.4	25.6	44.2	50.2
Auto-head$_L$	COCO	3.34	6.93	34.37	40.2	60.6	44.2	25.3	44.0	51.2
Auto-FPN	COCO	**1.61**$^{-52\%}$	**6.93**$^{-52\%}$	**32.64**$^{-22\%}$	**40.5**$^{+1.9}$	**61.5**$^{+1.1}$	**43.8**$^{+1.8}$	**25.6**$^{+1.9}$	**44.9**$^{+2.5}$	**51.0**$^{+1.8}$
Auto-FPN	BDD	1.61	5.7	31.41	39.2	60.7	42.3	24.1	43.4	49.9
Auto-FPN	VOC	1.83	5.39	31.32	38.9	60.4	41.9	23.3	43.0	49.6

Dataset	method	Search On	Params (neck)/M	Params (head)/M	Params (total)/M	mAP
PASCAL VOC	FPN	-	3.34	14.00	41.44	79.7
	Auto-fusion$_S$	VOC	1.54	14.00	39.65	80.7
	Auto-fusion$_M$	VOC	1.75	14.00	39.86	82.2
	Auto-fusion$_L$	VOC	1.83	14.00	39.93	82.7
	Auto-head$_S$	VOC	3.34	2.54	29.98	80.5
	Auto-head$_M$	VOC	3.34	5.24	32.69	81.3
	Auto-head$_L$	VOC	3.34	5.77	33.21	81.2
	Auto-FPN	VOC	**1.83**$^{-45\%}$	**5.24**$^{-63\%}$	**31.17**$^{-25\%}$	**81.8**$^{+2.1}$
	Auto-FPN	COCO	1.61	6.77	32.49	81.3
	Auto-FPN	BDD	1.61	5.54	31.26	81.4
BDD	FPN	-	3.34	13.95	41.39	33.0
	Auto-fusion$_S$	BDD	1.32	13.95	39.37	33.8
	Auto-fusion$_M$	BDD	1.61	13.95	39.66	33.8
	Auto-fusion$_L$	BDD	2.04	13.95	40.09	33.9
	Auto-head$_S$	BDD	3.34	1.44	28.88	33.8
	Auto-head$_M$	BDD	3.34	5.52	32.96	34.0
	Auto-head$_L$	BDD	3.34	6.34	33.78	33.9
	Auto-FPN	BDD	**1.61**$^{-52\%}$	**5.52**$^{-60\%}$	**31.23**$^{-25\%}$	**33.9**$^{+0.9}$
	Auto-FPN	COCO	1.61	6.75	32.46	33.7
	Auto-FPN	VOC	1.83	5.21	31.14	33.3

NAS baseline 比较结果

最后文中还给出了不同搜索策略的结果对比，可以看到基于梯度下降的效果还是很不错的。

	Search Method	No. arch searched	Search time (GPU days)	Average/Best mAP of searched arch
PASCAL VOC	Random	20	~ 18.3	76.4/80.3
	Evolutionary	60	~ 20.0	81.1
	SNAS[58]	1	~ 0.8	81.0
	Auto-FPN	1	**~ 0.8**	**81.8**
MS-COCO	Random	10	~ 130.0	36.4/38.2
	Evolutionary	30	~ 68.3	38.6
	SNAS[58]	1	~ 16.0	37.9
	Auto-FPN	1	**~ 16.0**	**40.5**

作者的一些 trick：

1) 先训练初始化结构一段时间，然后再开始搜索网络结构：Starting optimizing architecture parameters in the middle of training can improve the results by 1%;

2) 在搜索阶段固定 backbone 参数：Freezing the backbone parameters during searching not only accelerates the training but also improves the performance;

3) 搜错阶段不要用 BN：Searching with BN will decrease the performance by 1.7%;

4) 在搜索 head 结构时，最好使用训练好的 neck 结构：During searching for the head, loading the pretrained neck will boost the performance by 2.9%;

5) 大模型对 neck 部分性能有些许提升，但是对 head 好像并没有：Without resource constraints, our method becomes larger with only a small improvement in neck but no improvement in head.

总结：

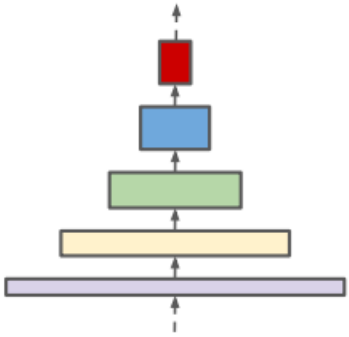
Auto-FPN 主要针对目标检测的两个更新：Auto-fusion 和 Auto-head。Auto-fusion 针对 FPN 的特征融合改进，即任意 N 层 level feature 特征融合,主要通过空洞卷积+rate, skip connection, depthwise-separable conv 以及上采样/下采样实现特征分辨率对齐和融合。Auto-head 采用 split-transform-merge 策略，search space 是 input nodes 和 intermediate nodes。

SpineNet

原文：
http://openaccess.thecvf.com/content_CVPR_2020/papers/Du_SpineNet_Learning_Scale-Permuted_Backbone_for_Recognition_and_Localization_CVPR_2020_paper.pdf

Motivation

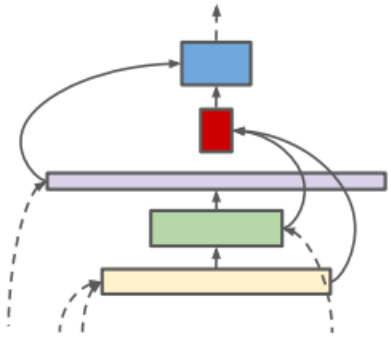
传统的神经网络在中间层计算时通常进行各种卷积操作以进行特征提取，一般是通过 maxpooling 或者 stride=2 的卷积对输入分辨率进行缩小。进行这样的操作可以使得输出特征具有更好的语义信息，从而可以更好的编码输入图像，取得较好的分类效果。但相同的 backbone 在目标检测问题中往往效果不好，尤其是对较小的目标来说。因为在目标检测中定位也同样重要，大量的细节信息可能在降采样的过程中丢失。在分类任务中我们采用了 encoder-decoder 架构，通过建立解码器恢复特征分辨率。但是应用于目标检测问题时仍然效果不好。文章提出了一种新的尺度可变的 backbone：SpineNet 以解决上述问题。



An example of scale-decreased network

Method

文中提出的尺度可变模型对传统的 backbone 架构主要做了两点改进：一是尺度可变，使其 feature map 的尺度可以任意放大或缩小，而与深度无关；二是引入多尺度跨层连接，以促进多尺度特征融合。改进后的 backbone 如下图：



Scale-permuted network

该 backbone 由一个固定的 stem network 和一个 scale-permuted network 构成。其中 stem network 为一个传统的尺度下降架构。而 scale-permuted network 由N个$blocks$构成，每个$block$属于某个特征层L_i。受 NAS-FPN 启发，文章采用L_3到L_7的 5 个$blocks$作为输出以及一个1×1卷积与每个输出块相连来得到同维度的多尺度特征。

NAS

在确定网络的架构模型后，文章采用 NAS（Neural Architecture Search，神经网络架构搜索）来确定具体的网络结构。

首先需要确定的是块的排序，因为一个$block$只能连接到顺序较低的$parent\ blocks$。文章通过分别排列中间块和输出块来定义排序的搜索空间。从我们构造网络可以得出这样的搜

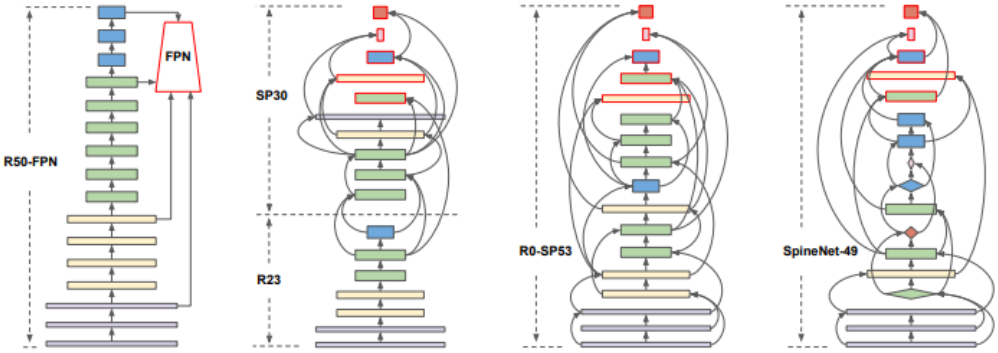
索空间为$(N-5)!\,5!$。

接下来需要搜索块之间的连接。定义每个块有两个输入，输入可以来自任何一个顺序较低的父块或者来自$stem\ network$。这样搜索空间的大小为$\prod_{i=m}^{N+m-1} C_2^i$，$m$为 stem network 中可选块的数量。

第三个我们搜索块的层级和种类。每个块可以按$\{-1,0,1,2\}$调整层级，使得搜索空间大小为4^{N-5}。另外每个块可以选择两种种类$\{bottleneck\ block, residual\ block\}$。搜索空间大小为$2^N$。

在构建 scale-permuted 架构时我们按照 ResNet-50 的架构构建。我们用 ResNet-50 的 bottleneck blocks 作为我们搜索空间中的可选特征块。另外也将 ResNet 中L_5中的一个 block 用L_6和L_7中的一个 block 替代。

构建的网络种类如下图所示：

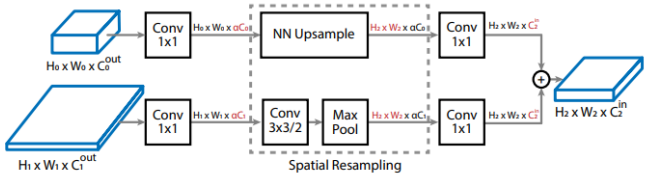


(a) R50-FPN @37.8% AP　(b) R23-SP30 @39.6% AP　(c) R0-SP53 @40.7% AP　(d) SpineNet-49 @40.8% AP

其中图（a）为 R50-FPN 模型。图（b）为剩余 7 个 ResNet 中的 7 个 block，以及 scale-permuted network 中的 10 个 block。图（c）为全部为 scale-permuted network 中的 block。图（d）为学习了额外的 block adjustment 后的 SpineNet-49。

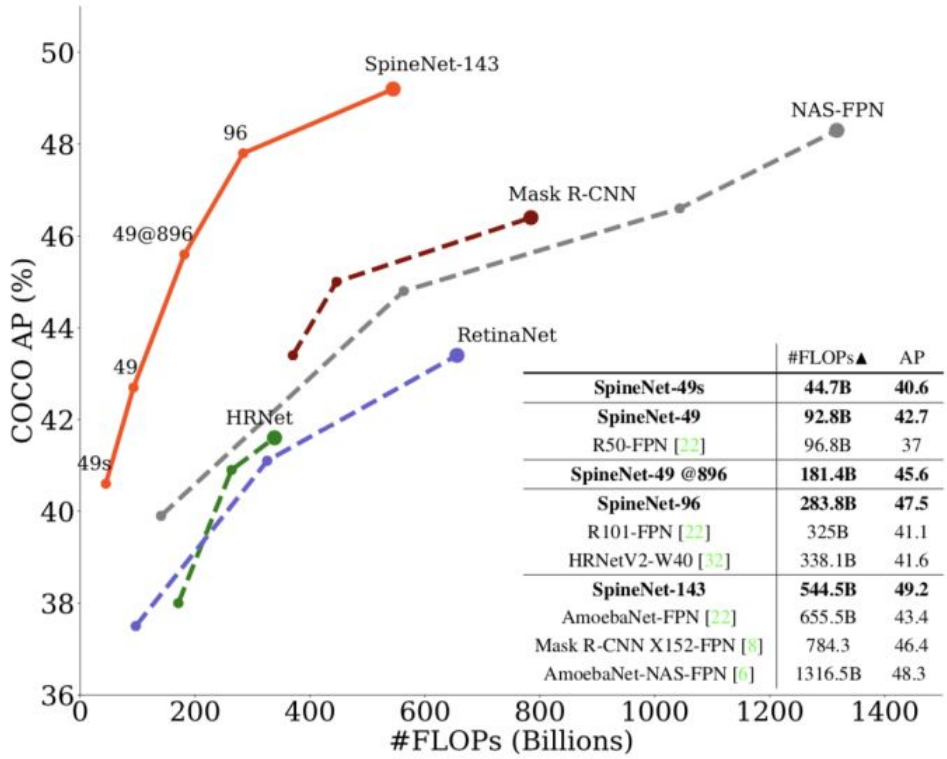
Resampling in Cross-scale Connections

我们网络构建时的难点在于父块和子块之间的特征尺寸和分辨率可能不同。为了解决这个问题，文章中添加了一个重采样机制以统一特征尺寸和分辨率。这里用C来表示3×3卷积层特征尺寸。在 bottleneck block 中，$C^{in}=C^{out}=4C$，在 residual block 中，$C^{in}=C^{out}=C$。引用一个尺寸因子α来调整来自父块的输出C^{out}为αC。然后我们用一个最近邻插值进行上采样或者一个 stride 为 2 的3×3卷积来下采样来匹配分配率。最终用一个1×1卷积来使αC匹配C^{in}。类似 FPN，我们将两个重新采样的输入特征图用 elemental-wise 合并。



Result

在 COCO 数据集上的一级目标检测上，对比 ResNet-FPN 不仅 AP 上提高了 6%，并且减少了 60%的算力需求。从实验结果看，SpineNet 比 NAS-FPN 还能取得进一步的提升。



	#FLOPs▲	AP
SpineNet-49s	**44.7B**	**40.6**
SpineNet-49	**92.8B**	**42.7**
R50-FPN [22]	96.8B	37
SpineNet-49 @896	**181.4B**	**45.6**
SpineNet-96	**283.8B**	**47.5**
R101-FPN [22]	325B	41.1
HRNetV2-W40 [32]	338.1B	41.6
SpineNet-143	**544.5B**	**49.2**
AmoebaNet-FPN [22]	655.5B	43.4
Mask R-CNN X152-FPN [8]	784.3	46.4
AmoebaNet-NAS-FPN [6]	1316.5B	48.3

并且值得一提的是，在 COCO 数据集检测任务上搜索出来的 SpineNet 进行泛化性非常不错，在检测、分割以及分类都有不错的表现。在 iNaturalist-2017 这种细粒度分类任务上更是提升近 6 个点，这主要是因为尺度多变的网络结构可以更好的保留细节信息。

总结

本文提出的主要贡献为：

构建了一个新的 backbone，其尺度可变，使其 feature map 的尺度可以任意放大或缩小，而与深度无关，并且引入多尺度跨层连接，以促进多尺度特征融合。对传统的骨干结构进行了创新。

通过引入 NAS 对 block 进行选择，使骨干结构更加合理，无需依靠人工进行先验设计。

Hit-detector

Motivation

NAS（神经网络架构搜索）在图片分类任务上取得了很好的成绩。一些最近的工作也成功搜索得到了一些目标检测中的 backbone 和特征融合层。但是这些工作都只关注了目标检测任务中的特定组成成分，而其他成分均根据先验设计。作者认为这种 NAS+人工设计的方式会限制网络的性能本文提供了一种 hierarchical trinity search framework，可以一次搜索目标检测中的三种组成（backbone，neck and head）。

关键概念：

- **Backbone**：目标检测中用到的 backbone，比如 ResNet 和 ResNeXt，通常为分类任务人工设计的网络。通常检测中比例最大的参数来自 backbone。例如 FPN 的 backbone，

ResNet-101，占到了 71%的参数比例。

- **Neck**：neck 连接着 backbone，起到多层特征融合的作用。利用网络内的特征金字塔来近似不同的接受场可以帮助检测器更好的定位。之前用于特征融合的 neck 架构均为人工设计，但是 NAS 可以搜索更好的连接方式实现不同尺度的特征融合。
- **RPN**（region proposal network）：一个典型的 RPN 是一个卷积层加上两个全连接层，用来选取候选区域和生成边框。本文沿用这种设计，不做搜索。
- **Head**：head 用来完成分类和定位任务。检测器通常拥有较大的 head 网络。例如 Faster R-CNN 利用了 ResNet 中的 5 层，FPN 利用了两个全连接层（全检测器 34%的参数量）。这样的设计实际上是低效的。

Method

先前将 NAS 应用于目标检测任务的公式可以概括为：

$$\alpha^* = \underset{\alpha \in A}{\operatorname{argmin}} f(\alpha) = \underset{\alpha \in A}{\operatorname{argmin}} L_{val}^{\det}\left(\alpha, w^*(\alpha)\right)$$

$$= \underset{\alpha \in A}{\operatorname{argmin}} L_{val}^{\det}(\alpha, \underset{w}{\operatorname{argmin}} L_{train}^{\det}(\alpha, w))$$

NAS 的优化目标即搜索一个架构$\alpha \in A$使得在权重w_α^*下，检验损失$L_{val}^{\det}$最小。

在本文的搜索网络中，作者将搜索空间扩大到 backbone，neck，head 三种成分，对原式进行了扩展，所以原式变为：

$$\alpha^*, \beta^*, \gamma^* = \underset{\alpha,\beta,\gamma}{\operatorname{argmin}} L_{val}^{\det}\left(\alpha, \beta, \gamma, w^*(\alpha, \beta, \gamma)\right)$$

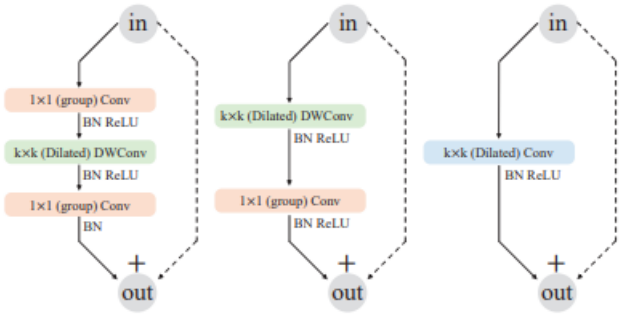
$$= \underset{\alpha,\beta,\gamma}{\operatorname{argmin}} L_{val}^{\det}(\alpha, \beta, \gamma, \underset{w}{\operatorname{argmin}} L_{train}^{\det}(\alpha, \beta, \gamma, w))$$

$$\alpha \in A_b, \beta \in A_n, \gamma \in A_h$$

其中A_b, A_n, A_h分别为 backbone，neck，head 的搜索空间。这样我们便可一次搜索得到三个结果。整体的搜索示意如图所示：

Hit-Detector 搜索架构示意图

在构建搜索空间时，借用 FBNet 的思想，构建不同的可选操作块。Inverted Residual block 包括一个1×1卷积，一个$k \times k$的 depthwise 卷积，另一个1×1卷积，再加上扩展因子 e。Separable block 包括一个$k \times k$的 depthwise 卷积和一个1×1卷积。如果输出的分辨率和输入的一样，用一个 skip connection 来把两个相加。



(a) Inverted Residual block (b) Separable block (c) Convolution block

搜索空间中的块结构

由于过大的搜索空间对 NAS 的搜索时间和内存要求过高，因此本文在大搜索空间中，分别为 backbone，neck，head 选取子搜索空间，以减少搜索时间和内存损耗。以 backbone 为例，设 backbone 共有L层，$\alpha_{l,i}$表示第l层的第i个操作所得分数，把α叫做结构化参数。在网络刚开始搜索时，结构化参数α的维度为$L \times N$，其中N为搜索空间的大小，默认值为 32。随着网络的不断搜索、训练，不同层的不同操作会得到不同的分数$\alpha_{l,i}$。将同一层，不同操作的分数进行过排序，将分数低的操作从搜索空间中删除。不断重复这个过程，直到结构化参数的维度降为$L \times N_b$。实验中我们令$N_b = N_n = N_h$

为了能够更加容易的将分数低的操作从搜索空间中删除（让同一层、不同操作的得分有更明显的差别），文章将结构化参数 a 进行了稀疏正则化：

$$\min_{\alpha} f(\alpha) + \mu \min_{i}(\sqrt{\sum_{l=1}^{L} \alpha_{l,i}^2}$$

其中μ为一个 trade-off 的超参。

接下来为端到端的搜索网络结构。其步骤和 DARTS 基本一致。仍以搜索 backbone 为例，第l层的输出为：

$$x_l = \sum_{o \in O_b} \frac{\exp(\alpha_l^o)}{\sum_{o' \in O_b} \exp(\alpha_l^{o'})} o(x_{l-1})$$

其中α为结构化参数，O_b表示 backbone 的搜索空间，o表示当前 NAS 选择的操作。Neck 和 head 的操作与 backbone 相同。由于结构化参数的存在，所以整个网络都是可微的，因此可以对网络进行端到端的搜索。

最后，本文为了防止搜索时间过长，搜索出的网络算力过大，对目标函数做了进一步优化。优化方式与 Proxylessnas 类似，给目标函数添加一个和算力有关的正则项：

$$\min_{\alpha,\beta,\gamma} L_{val}^{det}((\alpha, \beta, \gamma, w^*(\alpha, \beta, \gamma)) + \lambda(C(\alpha) + C(\beta) + C(\gamma)))$$

$$C(\alpha) = \sum_{l} \sum_{o \in O_b} \alpha_l^o FLOPs(o, l)$$

其中λ为平衡网络精度和算力的超参，$C(\alpha)$表示 backbone 的算力统计。

Result

文章将 Hit-Detector 与一些人工设计的网络，如使用不同 backbone 的 FPN，还有一些其他基于 NAS 实现的目标检测网络进行了对比实验。从表 2 结果上看，Hit-Detector 在较低参数量，较低算力的情况下，可以得到较好的 mAP

Table 2. Comparisons of the number of parameters, FLOPs and mAP on COCO minival. The FLOPs is based on the 800×1200 input and 1000 proposals in region proposal network. ‡ means the 2x schedule in training.

model	modified B	modified N	modified H	# params (total)	# FLOPs (total)	mAP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
FPN [26]	-	-	-	41.76M	197.4B	36.2	58.0	39.1	21.3	40.0	46.1
MobileNetV2 [44]	✓			19.61M	116.94B	30.1	51.8	30.9	16.7	33.0	38.7
ResNeXt-101 [49]	✓			60.38M	273.3B	40.3	62.1	44.1	23.6	45.0	51.6
FBNet-C [48]	✓			21.40M	119.0B	35.1	57.4	37.2	19.3	38.3	46.7
DetNAS-1.3G [8]	✓		✓	28.45M	254.1B	40.2	61.5	43.6	23.3	42.5	53.8
NASFPN [13]		✓		68.86M	616.9B	38.9	59.3	42.3	22.3	42.8	49.8
DetNAS-1.3G + NASFPN	✓	✓	✓	55.29M	672.9B	39.4	59.6	42.1	23.7	43.2	50.4
NATS-C [37]	✓			41.76M	197.4B	38.4	61.0	41.2	22.5	41.8	50.4
Auto-FPN‡ [51]		✓	✓	32.64M	476.6B	40.5	61.5	43.8	**25.6**	44.9	51.0
Hit-Detector	✓	✓	✓	27.12M	272.3B	**41.4**	**62.4**	**45.9**	25.2	**45.0**	**54.1**

文章也将 Hit-Detector 与一些经典的模型在 COCO 验证集上进行对比，从表 3 可以看出，在相同测试分辨率下，仍然可以取得较好的 mAP。

Table 3. Comparisons of single-model results on COCO test-dev.

model	test size	mAP
R-FCN [9]	600/1000	32.1
Faster R-CNN [42]	600/1000	30.3
Deformable [10]	600/1000	34.5
FPN [26]	800/1200	36.2
Mask R-CNN [18]	800/1200	38.2
RetinaNet [27]	800/1200	39.1
Light head R-CNN [24]	800/1200	41.5
PANet [32]	800/1000	42.5
TridentNet [23]	800/1200	42.7
NAS-FPN [13]	1024/1024	44.2
Hit-Detector	800/1200	44.5

总结

该文章的亮点在于，提出了一种可以一次搜索目标检测中的三种组成的端到端的神经网络，搜索出网络的效果比之前单纯搜索一种成分，其他成分人工设计的网络要好。在搜索网络的过程中采用了构建子搜索空间的方法，搜索方式融合了 FBNet 和 DARTS。