Neural Architecture Search for Dense Prediction Tasks

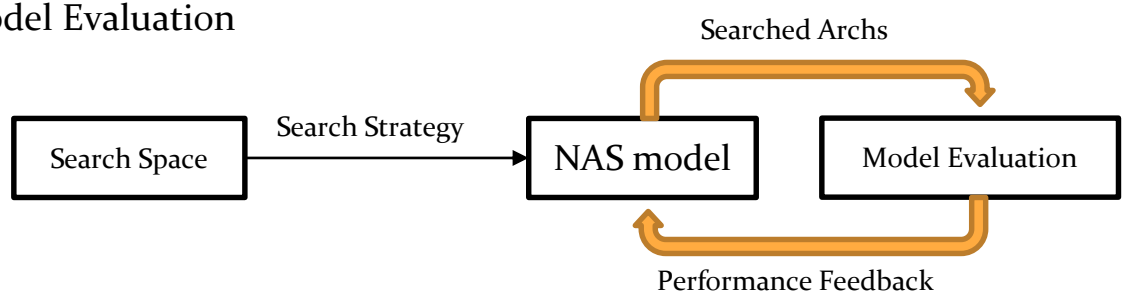
Chunhua Shen
The University of Adelaide

Fast Neural Architecture Search of Compact Segmentation Models via Auxiliary Cells, CVPR 2019, Nekrasov, Chen, Shen, Reid

- Manual design of architectures is a tedious task
- Instead, neural architecture search (NAS) methods automatically find “right” configurations for any given task
- Problem: Current reinforcement learning (RL)-based approaches for dense-per-pixel tasks (segmentation / depth / etc.) require hundreds or even thousands of GPU-days
- Our focus: how to train dense-per-pixel networks faster

Neural Architecture Search

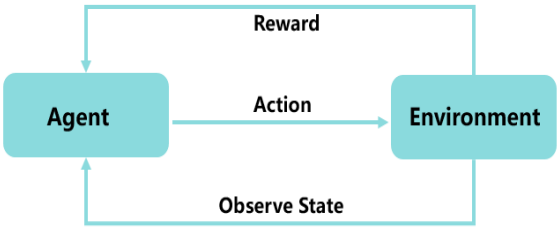
- Neural Architecture Search (NAS) can be divided as three parts：
  - Search Space
  - Search Strategy
  - Model Evaluation



Reinforcement Learning

- Reinforcement learning is learning what to do-how to map situations to actions-so as to maximize a numerical reward signal.

Elements	Brief Description
Policy	the learning agents' way of behaving at a given time
Reward Signal	the goal of a reinforcement learning problem
Value Function	judge what is good in the long run
Environment Model	something that mimics the behavior of the environment



RL-based NAS background

- A human designs the search space
- Step 2: An RL-agent (controller, RNN) outputs a string that describes a neural network configuration in the designed space
- The emitted configuration is trained and evaluated on the given task
- The reward (usually, validation loss) is recorded and the RL-agent is trained to produce configurations that maximise the expected reward
- Go to Step 2.

RL-based NAS background

To speed-up the search process, two solutions usually exist:

- To use thousands of GPUs and massively parallelise the whole search / training process,
- Or to use proxy datasets -- for example, instead of ImageNet, rely on CIFAR

Two questions with regards to that:

- What to do when there are no thousands of GPUs around?
- What is the proxy dataset for semantic segmentation on PASCAL VOC or CityScapes?

Our Approach

Instead, we are concentrating on methods to speed-up the training and evaluation of the emitted semantic segmentation models

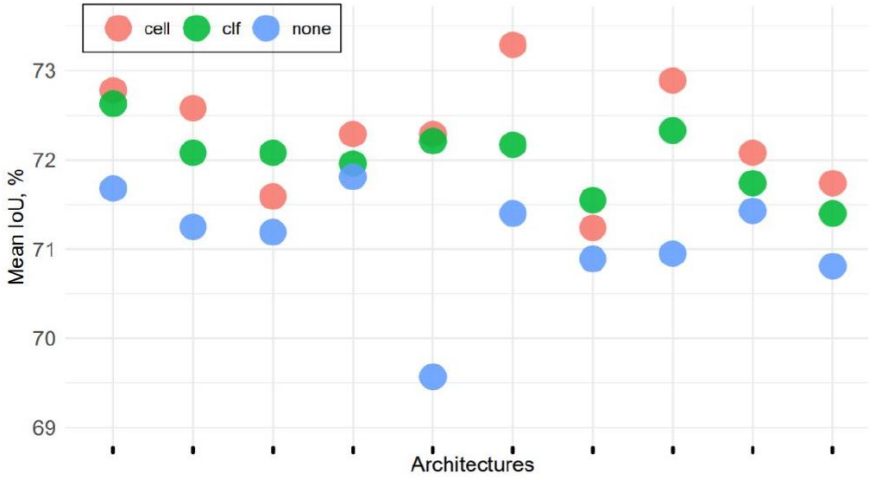
In particular:

1. We use two progressive stages with early stopping
   a. We start from a pre-trained backbone (MobileNet-v2) and search for the decoder only
   b. During the first stage, the encoder is fixed and its outputs are pre-computed -- only training the decoder
   c. If the performance after this stage is better than the running mean of previously seen results, we fine-tune end-to-end, otherwise -- move on to the next architecture

Our Approach

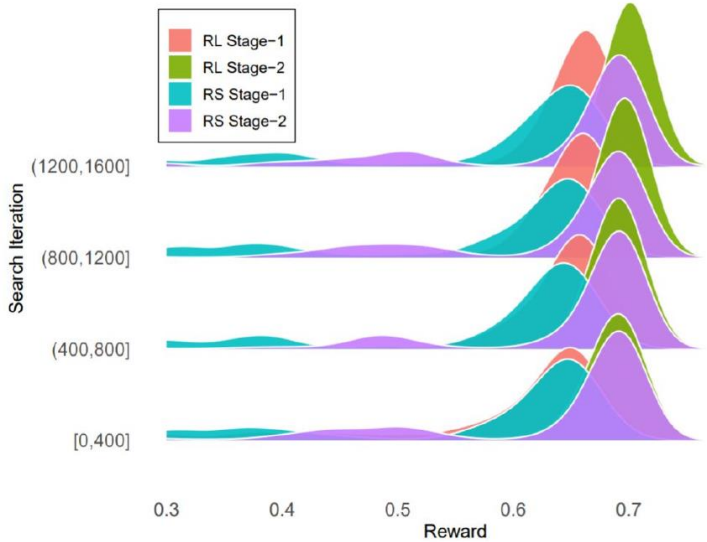
2. We use knowledge distillation and Polyak averaging during the first stage to speed-up the convergence of the decoder part

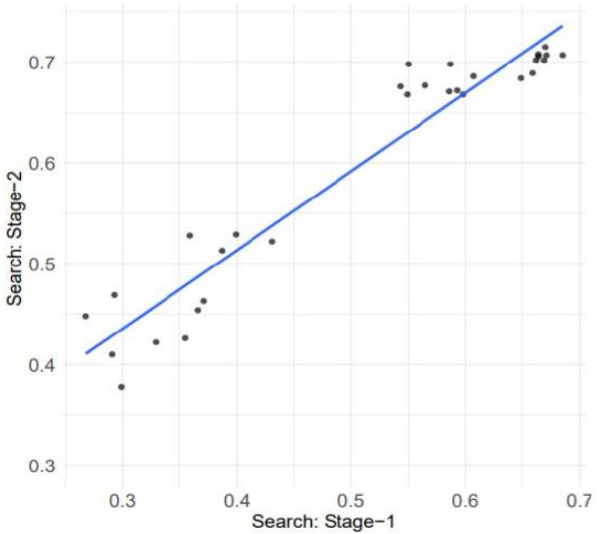
3. Finally, as our models are compact, we rely on intermediate supervision to further speed-up the convergence
    a. We found that over-parameterising auxiliary classifiers leads to even better results as opposed to single-layer

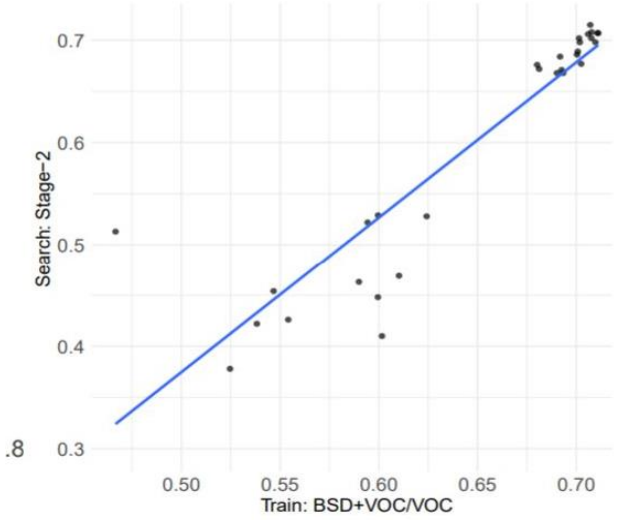


Results

- We found that our search rewards (geometric mean of mean IoU / freq.weighted-IoU and accuracy) correlate well with results after training for more epochs
- While comfortably beating random search







Results

- We tested 3 best architectures on PASCAL VOC and also studied their transfer to depth estimation and keypoint detection

Inference Results on VOC

Model	Val mIoU,%,	MAdds,B	Params,M	Output Res	Runtime,ms (JetsonTX2/1080Ti)	
DeepLab-v3-ASPP [32]	75.7	5.8	4.5	32×32	69.67±0.53	**8.09±0.53**
DeepLab-v3 [32]	75.9	8.73	**2.1**	64×64	122.07±0.58	11.35±0.43
RefineNet-LW [26]	76.2	9.3	3.3	128×128	144.85 ± 0.49	12.00±0.26
Ours (arch0)	**78.0**	4.47	2.6	128×128	109.36±0.39	14.86±0.31
Ours (arch1)	77.1	**2.95**	2.8	64×64	67.57±0.54	11.04±0.23
Ours (arch2)	77.3	3.47	2.9	64×64	**64.60±0.33**	8.86±0.26

Results

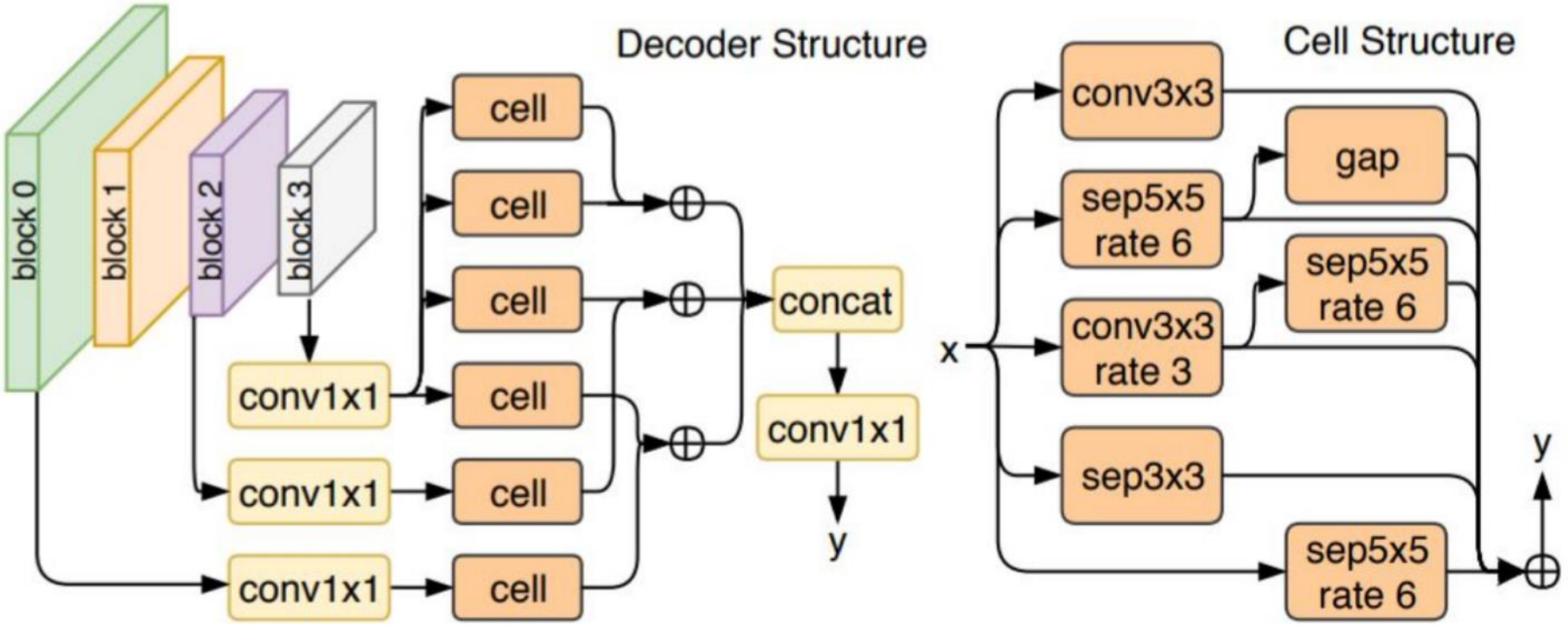
Transferring to Depth Estimation

	Ours				
	arch0	arch1	arch2	RF-LW [25]	CReaM [38]
RMSE (lin)	**0.523**	0.526	0.525	0.565	0.687
RMSE (log)	0.184	**0.183**	0.189	0.205	0.251
abs rel	0.136	**0.131**	0.140	0.149	0.190
sqr rel	0.089	**0.086**	0.093	0.105	–
$\delta < 1.25$	0.830	**0.832**	0.820	0.790	0.704
$\delta < 1.25^2$	0.967	**0.968**	0.966	0.955	0.917
$\delta < 1.25^3$	**0.992**	**0.992**	**0.992**	0.990	0.977
Parameters, M	2.6	2.8	2.9	3.0	**1.5**

Transferring to Keypoint Detection

	MPII		COCO		
Model	Mean@0.5	Mean@0.1	AP	AR	Params,M
DeepLab-v3+ [7]	86.6	31.7	0.668	0.700	5.8
ResNet-50 [43]	**88.5**	**33.9**	**0.704**	**0.763**	34.0
Ours (arch0)	86.5	31.4	0.658	0.691	**2.6**
Ours (arch1)	87.0	32.0	0.659	0.694	2.8
Ours (arch2)	87.1	31.8	0.659	0.693	2.9

Architecture Example



GAP = Global Average Pooling followed by conv1x1 and bilinear upsampling

Template-Based Architecture Search (https://arxiv.org/abs/1904.02365)

- In our CVPR paper, we used pre-trained encoder with around ~2M parameters and raised the total number to ~3M after the search
- Our focus in this work: how to search for segmentation architectures with <500K parameters?

Our Approach

Naive application of the methodology in the CVPR paper would lead us to large decision space: longer search process

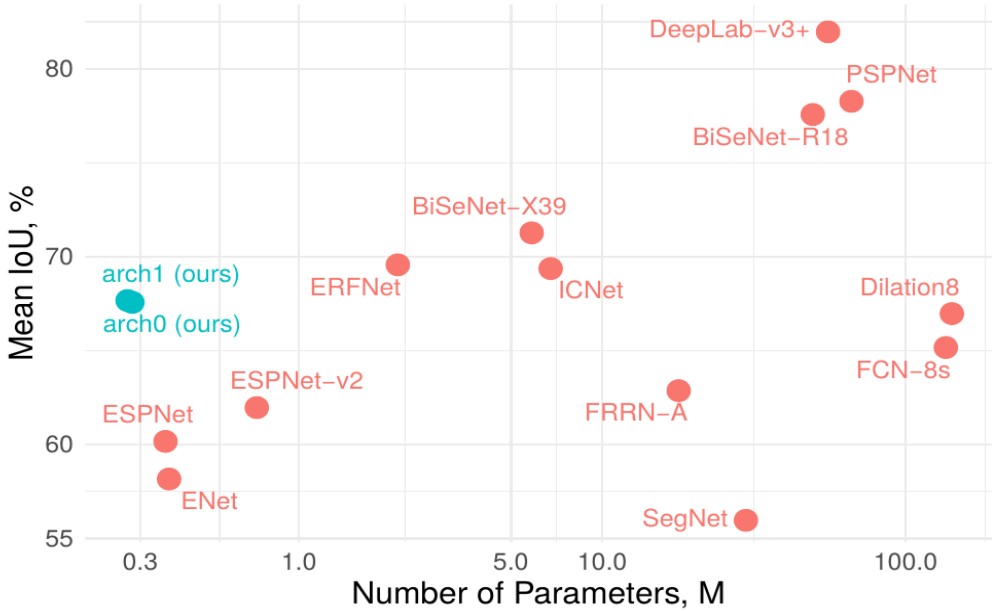
Instead,

- We factorise the search space into searching for shared templates (similar to cells) and locations where they should be used

- Additionally, to increase the capacity of the generated networks and adapt for segmentation needs, we add two decision nodes per layer:
  - Number of times the given template must be repeated (without weight sharing)
  - Stride of the first operation in the template

Results

- By using the factorised search space, we can search for extremely compact models by leveraging only 60K pre-trained parameters of MobileNet-v2
- We search using CityScapes



Results

CityScapes

Method	val mIoU,%	test mIoU,%	Params,M
ENet [17]	-	58.3	0.37
ESPNet [14]	61.4	60.3	0.36
ESPNet-v2 [15]	62.7	62.1	0.72
ICNet [28]	67.7	69.5	6.7
ERFNet [21]	71.5	69.7	2.1
BiSeNet [26]	**72.0**	**71.4**	5.8
Ours (arch0)	68.1	67.7[4]	**0.28**
Ours (arch1)	69.5	67.8[5]	**0.27**

Results

CamVid

Method	mIoU,%	Params,M
SegNet [1]	55.6	29.7
ESPNet [14]	55.6	0.36
DeepLab-LFOV [4]	61.6	37.3
†BiSeNet [26]	**65.6**	5.8
†ICNet [28]	**67.1**	6.7
Ours (arch0)	63.9	**0.28**
Ours (arch1)	63.2	**0.26**

Takeaway messages

- In order to leverage NAS for dense-per-pixel tasks it is not necessary to have 1000s of GPUs -- instead it is better to consider what is slowing us down in training

- Auxiliary over-parameterisation helps compact models to converge faster -- no use during inference

- Search for extremely compact models (<500K) does not result in FAST models -- we should consider setting up additional objectives for the controller

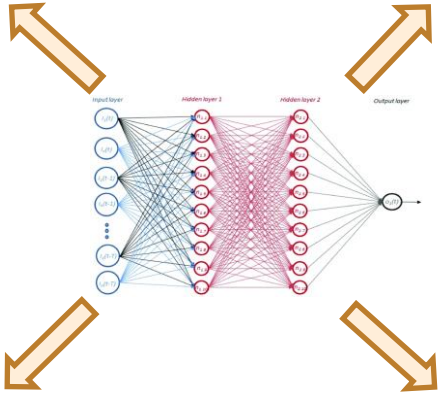
NAS-FCOS:
Fast Neural Architecture Search for Object Detection

Wang, Gao, Chen, Wang, Tian, Shen

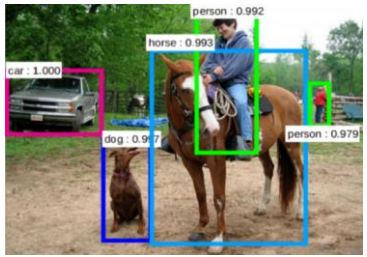
Background

- Amazing results in various visual tasks achieved by neural network-based approach:

Image Classification



Semantic Segmentation

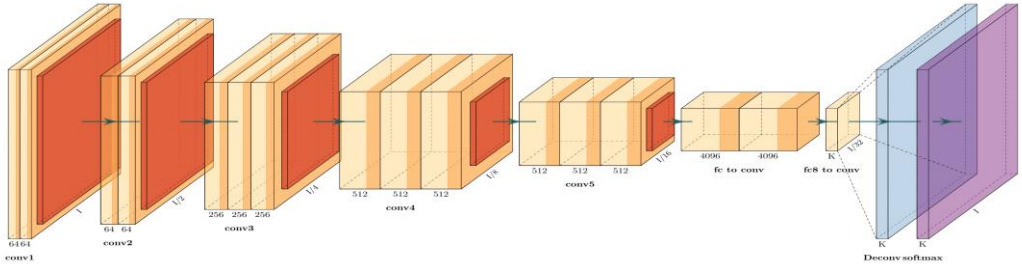


Object Detection

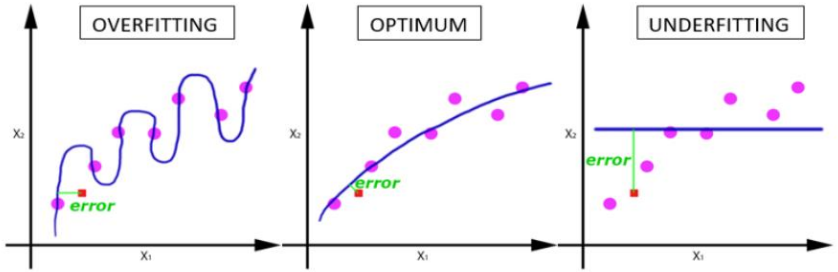
Pose Estimation

Problem and Target

Manual structural design



Adjusting parameters is difficult

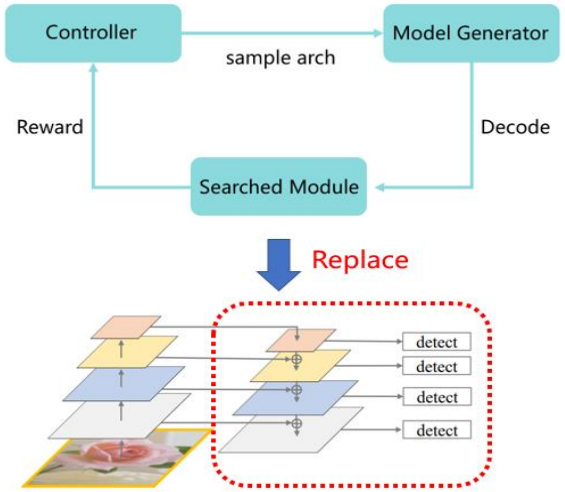


With the help of NAS, we hope to:

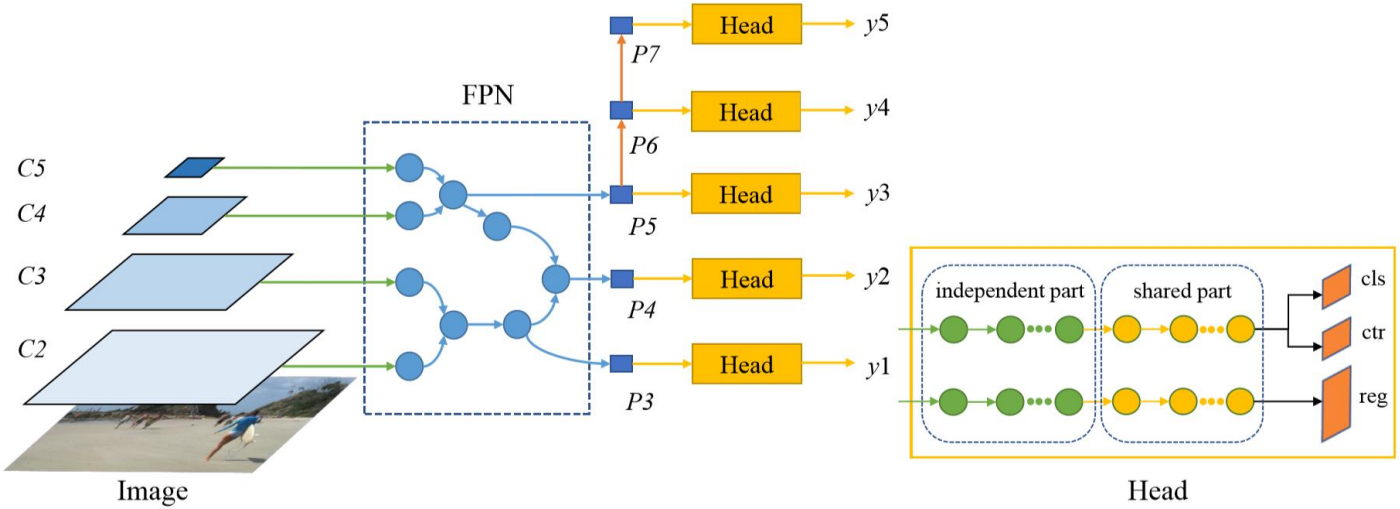
- break through the existing paradigm
- get rid of the tedious manual engineering

Search Framework

- The workflow of NAS-FCOS can be divided into three processes:
  - Sampling architecture from a search space following some search strategies
  - Evaluating performance of the sampled architecture
  - Updating the parameters based on the performance



Design Search Space



Our NAS-FCOS decoder consists of two sub networks, an FPN f and a set of prediction heads h which have shared structures. One notable difference with other FPN-based one-stage detectors is that our heads have partially shared weights. The number of layers to share is decided automatically by the search algorithm.

Improving Search Efficiency

- The key to NAS is to improve search efficiency. We propose the following ways to achieve the goal:
  - Proxy Dataset： Use PASCAL VOC as the proxy dataset, which contains 5715 training images with bounding box annotations.
  - New evaluate metric： Use negative loss sum as the reward instead of average precision.
  - Progressive search： Use a progressive search strategy rather than the joint search for both FPN and head, since the former requires less computing resources and time cost than the latter.

New evaluate metric

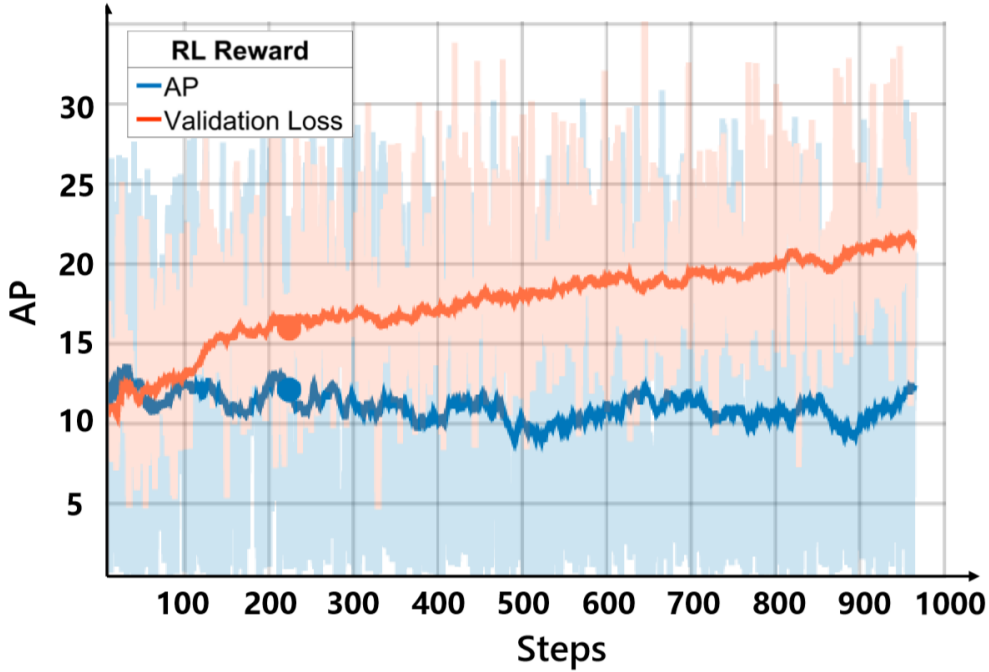
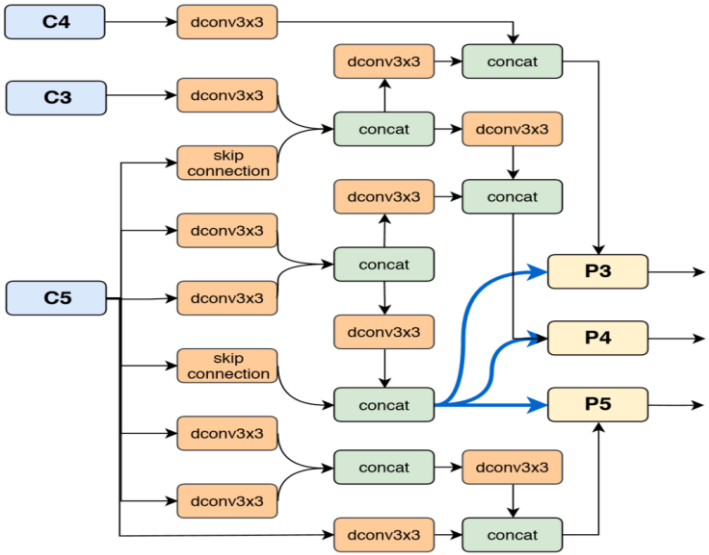


Figure 7: Comparison of two different RL reward designs. The vertical axis represents AP obtained from the proxy task on the validation dataset.

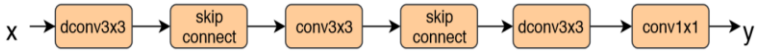
Search Results

- The controller identifies that deformable convolution and concatenation are the best performing operations for unary and aggregation respectively.
- Note that the discovered “dconv+1x1conv” structure achieves a better trade-off between accuracy and FLOPs than “3x3conv+3x3conv”.
- The searched decoder with channel 256 (@256) surpasses its FCOS counterpart by 1.5 to 3.5 points in AP under different backbones.
- NAS-FCOS model still achieves better performance (AP=38.9 with FPN search only, and AP=39.8 with both FPN and head search) than the Deform-FPN-FCOS model (AP=38.4).
- Searching for FPN brings slightly more benefits than searching head only. And our progressive search which combines both FPN and head achieves a better result.

Discovered Structures



Our discovered FPN structure. C2 is omitted from this figure since it is not chosen by this particular structure during the search process.



Our discovered Head structure.

Experiments

Decoder	Backbone	FLOPs (G)	Params (M)	AP
FPN-RetinaNet @256	MobileNetV2	133.4	11.3	30.8
FPN-FCOS @256	MobileNetV2	105.4	9.8	31.2
NAS-FCOS (ours) @128	MobileNetV2	**39.3**	**5.9**	32.0
NAS-FCOS (ours) @128-256	MobileNetV2	95.6	9.9	33.8
NAS-FCOS (ours) @256	MobileNetV2	121.8	16.1	**34.7**
FPN-RetinaNet @256	R-50	198.0	33.6	36.1
FPN-FCOS @256	R-50	169.9	32.0	37.4
NAS-FCOS (ours) @128	R-50	**104.0**	**27.8**	37.9
NAS-FCOS (ours) @128-256	R-50	160.4	31.8	39.1
NAS-FCOS (ours) @256	R-50	189.6	38.4	**39.8**
FPN-RetinaNet @256	R-101	262.4	52.5	37.8
FPN-FCOS @256	R-101	**234.3**	**50.9**	41.5
NAS-FCOS (ours) @256	R-101	254.0	57.3	**43.0**
FPN-FCOS @256	X-64x4d-101	371.2	89.6	43.2
NAS-FCOS (ours) @128-256	X-64x4d-101	**361.6**	**89.4**	**44.5**
FPN-FCOS @256 w/improvements	X-64x4d-101	371.2	89.6	44.7
NAS-FCOS (ours) @128-256 w/improvements	X-64x4d-101	**361.6**	**89.4**	**46.1**

Results on test-dev set of COCO after full training. FLOPs and parameters are being measured on 1088×800. For our NAS-FCOS, @128 and @256 means that the decoder channel width is 128 and 256 respectively. @128-256 is the decoder with 128 FPN width and 256 head width.

Experiments

Arch	FLOPs (G)	Search Cost (GPU-day)	Searched Archs	AP
NAS-FPN @256 R-50	>325.0	333×#TPUs	17000	<38.0
NAS-FPN 7@256 R-50	1125.5	333×#TPUs	17000	44.8
DetNAS-FPN-Faster	-	44	2200	40.0
DetNAS-RetinaNet	-	44	2200	33.3
NAS-FCOS (ours) @256 R-50	**189.6**	**28**	3000	39.8
NAS-FCOS (ours) @128-256 X-64x4d-101	361.6	**28**	3000	**46.1**

Comparison with other NAS methods. For NAS-FPN, the input size is 1280 × 1280 and the search cost should be timed by their number of TPUs used to train each architecture. Note that the FLOPs and AP of NAS-FPN @256 here are from Figure 11 in NAS-FPN(Ghiasietal.2019), and NAS-FPN 7@256 stacks the searched FPN structure 7 times.

Experiments

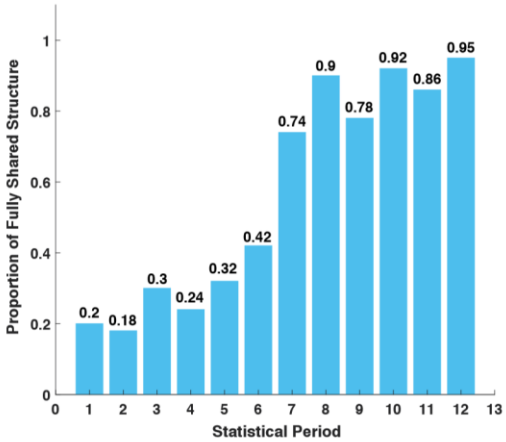


Figure 5: Trend graph of head weight sharing during search. The coordinates in the horizontal axis represent the number of the statistical period. A period consists of 50 head structures. The vertical axis represents the proportion of heads that fully share weights in 50 structures.

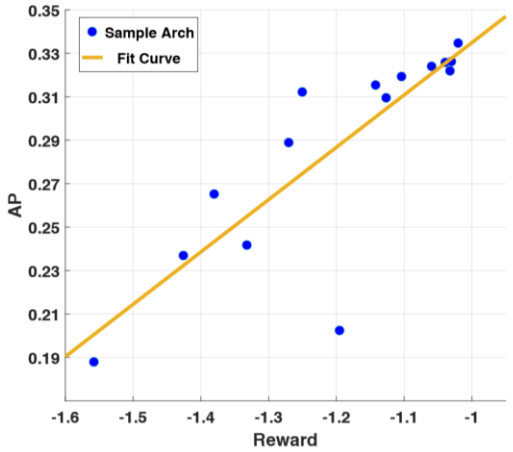


Figure 6: Correlation between the search reward obtained on the VOC meta-val dataset and the AP evaluated on COCO-val.

Messages

- Neural Architecture Search can be used to further design and optimize object detection networks.
- Carefully designed proxy tasks, search strategies and model evaluation metrics are necessary.
- The discovered NAS-FCOS models are efficient and flexible with various backbone architectures.

IR-NAS:
Neural Architecture Search for Low-level Image Restoration

Zhang, Chen, Li, Shen

IR-NAS

- IR-NAS is one of the first efforts towards employing NAS algorithm to automatically design effective neural network architectures for low-level image restoration tasks

- IR-NAS is able to search for both inner cell structures and outer layer widths. It takes only 6 hours to search on a single GPU and takes one third of the memory of Auto-Deeplab (Liu et al. 2019) to search for the same structure.

- The proposed IR-NAS is applied to such tasks: image denoising and image deraining. The architectures found by IR-NAS outperform SOTA algorithms with less parameters and faster speed.

Liu et al. 2019. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In CVPR, 82–92.

IR-NAS

Following (Liu et al. 2019; Cai et al. 2019), we employ gradient-based architecture search strategy in our IR-NAS and we search a computation cell as basic block then build the final architecture by stacking the searched block with different widths.

Differing from previous methods, IR-NAS has a more flexible search space and it is able to search both the cell structures and widths.

Zhu et al. 2019. Darts: Differentiable architecture search. In ICLR.
Cai et al. 2019. Proxylessnas: Direct neural architecture search on target task and hardware. In ICLR

Cell architecture search

For cell architecture search, we build a super cell that integrate all possible layer types and consists of N nodes, then derive a compact cell from the build super cell according to the learned continuous weights α.

We denote the super cell in layer l as Cl, which takes outputs of previous cell and the cell before previous cell as inputs and outputs one tensor hl . As shown in Figure 1.

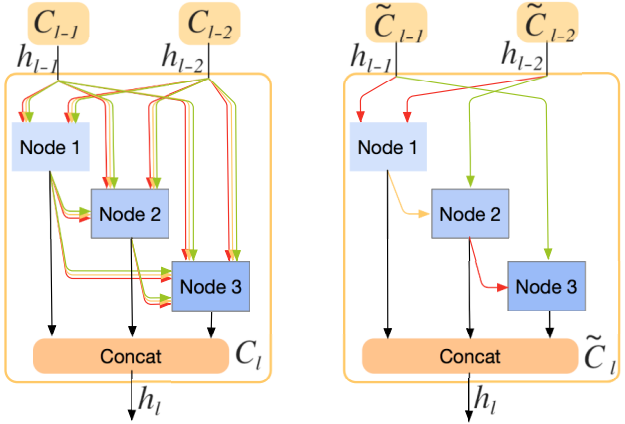


Figure 1. Cell architecture search. Left: super cell. Right: the cell architecture search result.

Cell architecture search

The output of the i th node in Cl is calculated as follows:

$$x_{l,i} = \sum_{x_j \in I_{l,i}} O_{j \to i}(x_j),$$

$$O_{j \to i}(x_j) = \sum_{k=1}^{S} \alpha_{j \to i}^{k} O^{k}(x_j),$$

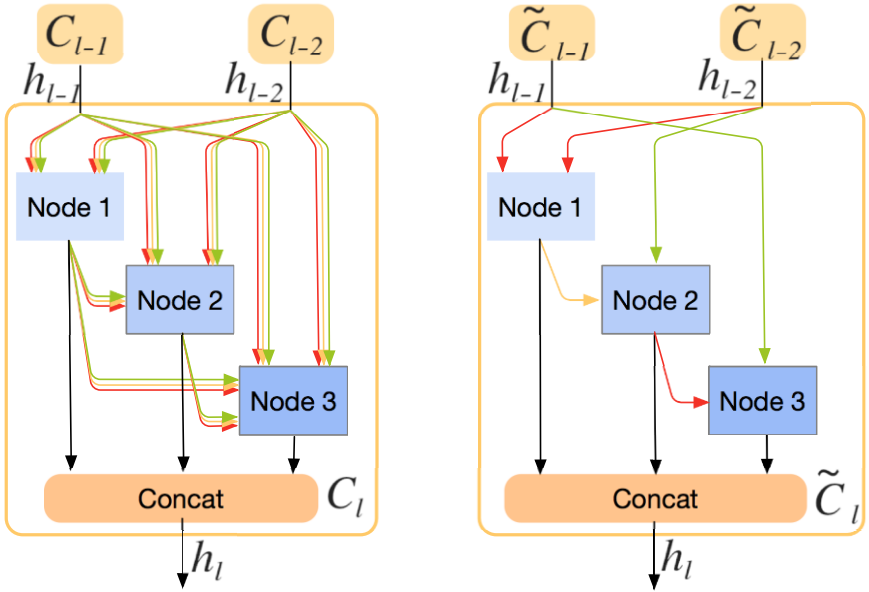


Figure 1. Cell architecture search. Left: super cell. Right: the cell architecture search result.

Cell architecture search

:

- conv: 3X3 convolution
- sep: 3X3 separable convolution
- dil: 3X3 convolution with dilation factor 2
- def: 3X3 deformable convolution V2 (Zhu et al. 2019)
- skip: skip connection
- none: no connection and return zero

Zhu et al. 2019. Deformable convnets v2: More deformable, better results. In CVPR, 9308–9316.

Cell architecture search

The output of the super cell Cl can be expressed as :

$$h_l = Cell(h_{l-1}, h_{l-2})$$
$$= Concat\{x_{l,i} | i \in \{1, 2, \cdots, N\}\}$$

After the super cell is trained, for each node, we rank the corresponding inputs according to α values, then reserve the top two inputs and remove the rest to obtain the compact cell, as shown in the right of Figure 1

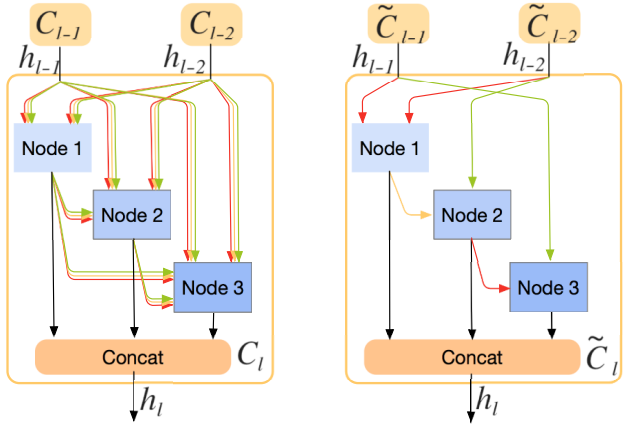


Figure 1. Cell architecture search. Left: super cell. Right: the cell architecture search result.

Cell width search

Similarly, we build a super net that contains several super cells with different widths in each layer. As illustrated in the left of Figure 2.

At each layer l, there are three cells with widths W , 2W and 4W , where W is the basic width. The output feature of each layer is:

$$h_l = \{h_l^0, h_l^1, h_l^2\},$$

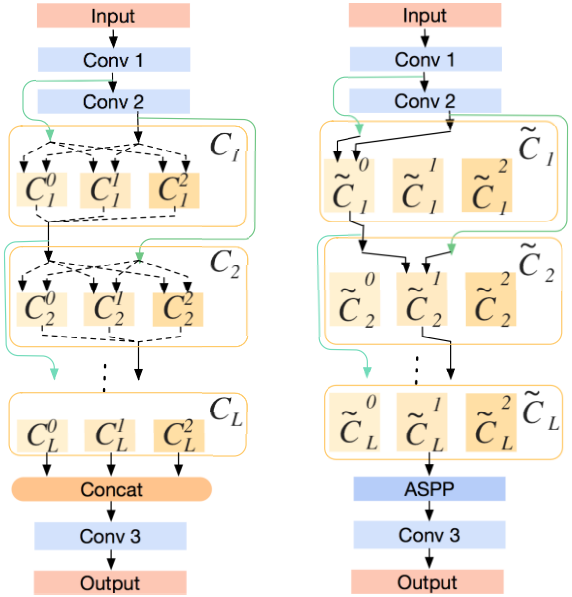


Figure 2. Cell width search. Left: a super net Right: the final architecture

Cell width search

After the super net is trained with gradient descent, we view the learned β values as probability, then use the Viterbi algorithm to select the path with the maximum probability as the final result.

In addition, an ASPP module is added to the end of the last Cell in the final architecture, as illustrated in the right of Figure 2.

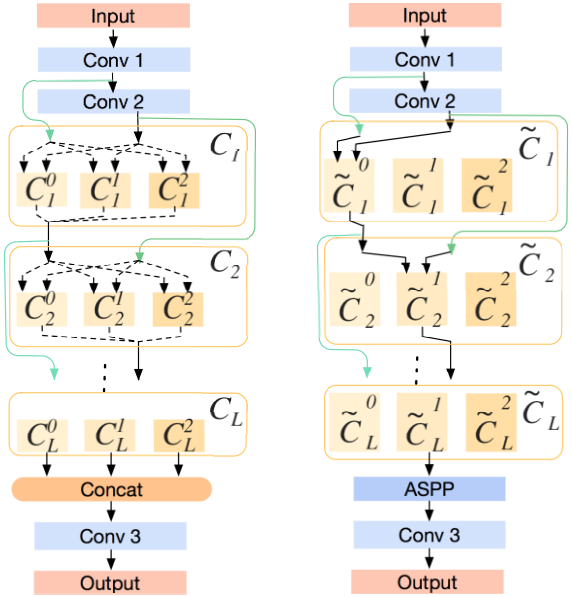


Figure 2. Cell width search. Left: a super net Right: the final architecture

Searching with gradient descent

The searching process is the optimization process. For image denoising and image de-raining, the two most widely used evaluation metrics are PSNR and SSIM (Wang et al. 2004). Inspired by this, we de- sign the following loss for optimizing super net:

$$loss = \|IRNAS(x) - y\|_2^2 + \lambda log\_ssim(IRNAS(x), y),$$

$$log\_ssim(x, y) = log_{10}(SSIM(x, y)^{-1})$$

Wang et al. 2004. Image quality assessment: from error visibility to structural similarity. IEEE Trans. Image Process. 13(4), 600–312.

Networks found by IR-NAS

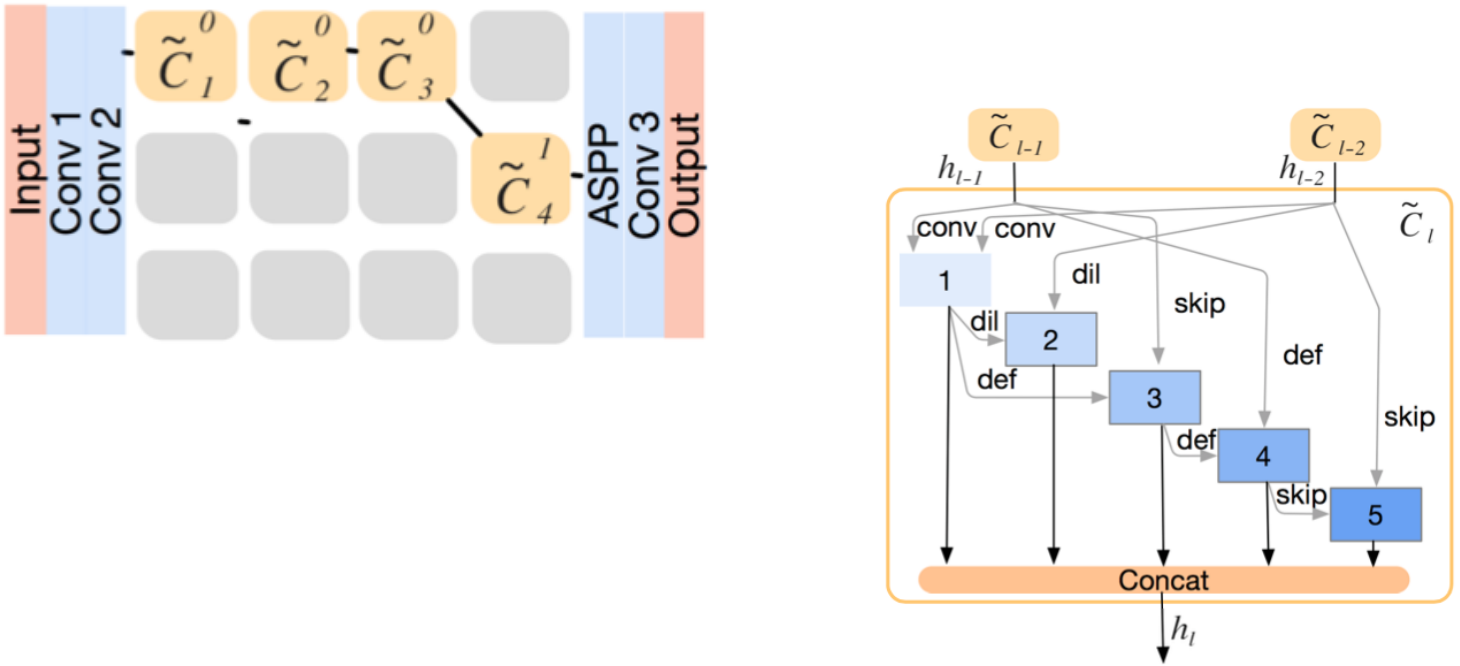


Figure 3. Left: the denoising network. The first one shows the out layer widths and the second one shows the inner cell architectures.

Experimental results on BSD500

Methods	# parameters (M)	time cost (s)	$\sigma = 30$		$\sigma = 50$		$\sigma = 70$	
			PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
BM3D (Dabov et al. 2007)	-	-	27.31	0.7755	25.06	0.6831	23.82	0.6240
WNNM (Gu et al. 2014)	-	-	27.48	0.7807	25.26	0.6928	23.95	0.3460
RED (Mao, Shen, and Yang 2016)	0.99	-	27.95	0.8056	25.75	0.7167	24.37	0.6551
MemNet (Tai et al. 2017)	4.32	-	28.04	0.8053	25.86	0.7202	24.53	0.6608
NLRN (Liu et al. 2018)	0.98	10411.49	28.15	**0.8423**	25.93	0.7214	24.58	0.6614
N3Net (Plötz and Roth 2018)	0.68	121.11	28.66	0.8220	26.50	0.7490	25.18	0.6960
IR-NAS	**0.63**	**83.25**	**29.14**	0.8403	**26.77**	**0.7635**	**25.48**	**0.7129**

Table 1: Denoising experiments. Comparisons with state-of-the-arts on BSD500. We show our results in the last row. Time cost means GPU-seconds cost to inference on the 200 images from the test set of BSD500 with one GTX 980 graphic card.

Figure 4: Denoising experiments on on BSD500.

What we are doing now

- Further improving the efficiency and flexibility of IR-NAS. Specifically, by learning the advantages of ProxylessNAS and Single-Path NAS, we will propose new search strategy and search space, where network depth is also included and the changing of outer layer widths is more flexible.
- Expanding the proposed IR-NAS algorithm to more image restoration tasks, such as super-resolution, dehazing and inpainting, etc.