Table 1: Detailed statistics of our NAS-Bench-MR. NAS-Bench-MR contains 4 datasets and 9 settings. Considering the diversity and complexity of real-world applications (*e.g.*different scales/input sizes of training data), we use a variety of challenging settings (*e.g.*, full resolution and training epochs) to ensure that the model is fully trained. For classification, we train models with different numbers of classes, numbers of training samples, and training epochs. For semantic segmentation, we train models under different input sizes. We also evaluated video action recognition models under two settings: trained from scratch and pretrained with ImageNet-50-1000. † denotes each sample contains multiple classes. ‡ denotes there are three basic classes (car, pedestrian, cyclist) in KITTI, while for each object we also regress its 3D location (XYZ), dimensions (WHL), and orientation (α). "N" – training from scratch. "Y" – training with the ImageNet-50-1000 pretrained model. We also give the mean, std, and the validation L1 loss (%) of the neural predictor under the main evaluation metric of each setting.

	Cls-50-1000	Cls-50-100	Cls-10-1000	Cls-10c	Seg	Seg-4x	3dDet	Video	Video-p
Dataset	ImNet-50-1000	ImNet-50-100	ImNet-10-1000	ImNet-50-1000	Cityscapes	resized Cityscapes	KITTI	HMDB51	HMDB51
Input Size	224 × 224	224 × 224	224 × 224	224 × 224	512 × 1024	128 × 256	432 × 496	112 × 112	112 × 112
Epochs	100	100	100	10	530	530	80	100	100
Number of classes	50	50	10	50	19	19	3‡	51	51
Number of training samples per class	1000	1000	1000	1000	2975†	2975†	3712†	≈ 70	≈ 70
Pretrained?	N	N	N	N	N	N	N	N	Y
Main metric	top1 Acc	top1 Acc	top1 Acc	top1 Acc	mIoU	mIoU	car-3D AP	top1 Acc	top1 Acc
Mean	82.21	48.59	85.70	57.07	72.58	62.21	76.36	19.56	30.88
Standard Deviation	2.64	3.69	1.51	4.51	5.33	3.37	4.70	3.69	4.65
Prediction Loss	0.56	1.07	0.83	1.05	0.75	0.88	0.85	1.07	1.26

APPENDIX – LEARNING VERSATILE NEURAL ARCHITECTURES BY PROPAGATING NETWORK CODES

Anonymous authors
Paper under double-blind review

A DETAILS OF NAS-BENCH-MR

In this section, we provide details of the datasets and settings used to build our NAS-Bench-MR [1]. We follow the common practice and realistic settings for training and evaluation in each task, making the scientific insights generated by our benchmark easier to generalize to real-world scenarios.

A.1 DATASETS AND SETTINGS

We randomly sample 2,500 structures from our network coding space, and train and evaluate these same architectures for each task and setting. Each architecture is represented by a 27-dimensional continuous-valued code. Tab. 2 shows the representation of each dimension of the code. For 9 different datasets and proxy tasks, we train a total of 22,500 models. See below for details.

ImageNet for Image Classification. The ILSVRC 2012 classification dataset (Deng et al., 2009) consists of 1,000 classes, with a number of 1.2 million training images and 50,000 validation images.

Considering the diversity of the number of classes and data scales in practical applications, and saving training time, we construct three subsets: ImageNet-50-1000, ImageNet-50-100, and ImageNet-10-1000, where the first number indicates the number of classes and the second one denotes the number of images per class.

In this work, we adopt an SGD optimizer with momentum 0.9 and weight decay 1e-4. The input size is 224 × 224. The initial learning rate is set to 0.1 with a total batch size of 160 on 2 Tesla V100 GPUs for 100 epochs, and decays by cosine annealing with a minimum learning rate of 0. We

[1] Project page: https://network-propagation.github.io

Table 2: Representations of our 27-dimensional coding.

Component	Codes		
	N_{blocks}	$N_{residual\ units}$	$N_{channels}$
Input layers			i_1, i_2
Stage 1	b_1	n_1^1	c_1^1
Stage 2	b_2	n_2^1, n_2^2	c_2^1, c_2^2
Stage 3	b_3	n_3^1, n_3^2, n_3^3	c_3^1, c_3^2, c_3^3
Stage 4	b_4	$n_4^1, n_4^2, n_4^3, n_4^4$	$c_4^1, c_4^2, c_4^3, c_4^4$
Output layer			o_1

adopt the basic data augmentation scheme to train the classification models, i.e., random resizing and cropping, and random horizontal flipping (flip ratio 0.5), and use single-crop for evaluation.

We report the top-1 and top-5 accuracies as the evaluation metric on all three benchmark datasets. We print the training log (top-1, top-5, loss, lr) every 20 iterations and evaluate the model every 5 epochs on the ImageNet validation set. We will release four checkpoints of 25, 50, 75, 100 epochs with the optimizer data and all the training logs for each model.

Cityscapes for Semantic Segmentation. The Cityscapes dataset (Cordts et al., 2016) contains high-quality pixel-level annotations of 5000 images with size 1024×2048 (2975, 500, and 1525 for the training, validation, and test sets respectively) and about 20000 coarsely annotated training images. Following the evaluation protocol (Cordts et al., 2016), 19 semantic labels are used for evaluation without considering the void label.

To study the effect of different image resolutions in segmentation, we conduct two benchmarks with the input size of 512×1024 and 128×256, respectively. For the small-resolution setting, we pre-resize the Cityscapes dataset to 256×512 before the data augmentation.

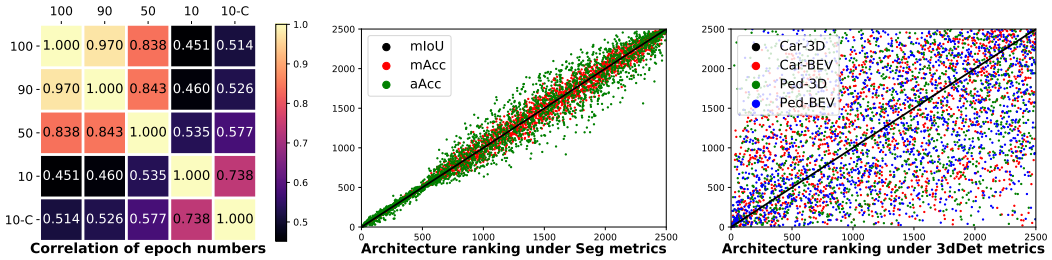
In this work, we use an SGD optimizer with momentum 0.9 and weight decay 4e-5. The initial learning rate is set to 0.1 with a total batch size of 64 on 8 Tesla V100 GPUs for 25000 iterations (about 537 epochs). Follows the common practice in (Zhao et al., 2017; Wang et al., 2020), the learning rate and momentum follow the "poly" scheduler with power 0.9 and a minimum learning rate of 1e-4. We use basic data augmentation, *i.e.*, random resizing and cropping, random horizontal flipping, and photometric distortion for training and single-crop testing with the test size of 1024×2048 and 256×512 respectively for two settings.

We report the mean Intersection over Union (mIoU), mean (macro-averaged) Accuracy (mAcc), and overall (micro-averaged) Accuracy (aAcc) as the evaluation metrics. We print the training log (mAcc, loss, lr) every 50 iterations and evaluate the model every 5000 iterations on the Cityscapes validation set. We will release five checkpoints of 5000, 10000, 15000, 20000, 25000 iterations with the optimizer data and all the training logs for each model.

KITTI for 3D Object Detection. The KITTI 3D object detection dataset (Geiger et al., 2012) is widely used for monocular and LiDAR-based 3D detection. It consists of 7,481 training images and 7,518 test images as well as the corresponding point clouds and the calibration parameters, comprising a total of 80,256 2D-3D labeled objects with three object classes: Car, Pedestrian, and Cyclist. Each 3D ground truth box is assigned to one out of three difficulty classes (easy, moderate, hard) according to the occlusion and truncation levels of objects.

In this work, we follow the train-val split (Chen et al., 2015), which contains 3,712 training and 3,769 validation images. The overall framework is based on Pointpillars (Lang et al., 2019). The input point points are projected into bird's-eye view (BEV) feature maps by a voxel feature encoder (VFE). The projected BEV feature maps (496×432) are then used as the input of our 2D network for 3D/BEV detection.

Following (Lang et al., 2019), we set, pillar resolution: 0.16m, max number of pillars: 12000, and max number of points per pillar: 100. We apply the same data augmentation, *i.e.*, random mirroring and flipping, global rotation and scaling, and global translation for 3D point clouds as in Pointpillar (Lang et al., 2019). We use the one-cycle scheduler with an initial learning rate of 2e-3, a minimum learning rate of 2e-4, and batch size 16 on 8 Tesla V100 GPUs for 80 epochs. We use an AdamW optimizer with momentum 0.9 and weight decay 1e-2. At inference time, axis-aligned non-maximum suppression (NMS) with an overlap threshold of 0.5 IoU is used for final selection.



(a) (b) (c)

Figure 1: (a) Spearman's rank correlation of the different number of epochs (checkpoints of 10, 50, 90, 100 epochs during training on ImageNet-50-1000). '10-C' denotes using the convergent learning rate for 10 epochs, *i.e.*, the proxy setting used in RegNet (Radosavovic et al., 2020). (b) Architecture rankings under the evaluation metrics of semantic segmentation (mIoU, mAcc, aAcc). (c) Architecture rankings under the evaluation metrics of 3D object detection (car/pedestrian 3D/bird's-eye view detection AP).

We report standard average precision (AP) on each class as the evaluation metric. We will release five checkpoints of 76, 77, 78, 79, 80 epochs with the optimizer data and detailed evaluation results (Car/Pedestrian/Cyclist easy/moderate/hard 3D/BEV/Orientation detection AP) of each checkpoint for each model. Due to the unstable training of the KITTI dataset, we provide the last five checkpoints for researchers to tune hyperparameters from different criteria, such as the single model best performance, the average best performance, and the last epoch best performance.

HMDB51 for Video recognition. We train video recognition models on the HMDB51 dataset (Kuehne et al., 2011), consisting of 6,766 videos with 51 categories. We use the first training and validation split composing of 3570 and 1530 videos for evaluation, respectively. The video containing less than 64 frames is filtered, resulting in 2649 samples for training.

Considering the difficulty of learning temporal information, in addition to training from scratch, we also conduct training using models of ImageNet-50-1000 as pretraining. To model the temporal information, we replace the last 2D convolutional layer before the final classification layer of HRNet and its counterparts with a 3D convolutional layer.

In this work, the input size is set to 112×112. We adopt an Adam optimizer with momentum 0.9 and weight decay 1e-5. The initial learning rate is set to 0.01 with a total batch size of 80 on 4 Tesla V100 GPUs for 100 epochs, and decays by cosine annealing with a minimum learning rate of 0. We adopt random resizing and cropping, random brightness 0.5, random contrast 0.5, random saturation 0.5, random hue 0.25, and random grayscale 0.3 as data augmentation.

We report the top-1 and top-5 accuracies as the evaluation metric on both two settings. We print the training log (top-1, top-5, loss, lr) every 10 iterations and evaluate the model every 10 epochs on the validation set. We will release the last checkpoint and all the training logs for each model.

A.2 Analysis of NAS-Bench-MR

We summarize detailed statistics of our NAS-Bench-MR in Tab. 1. We also give the mean, std, and the validation L1 loss (%) of our neural predictors under the main evaluation metric of each setting.

From Spearman's rank correlation of nine benchmarks in our main paper, we observe that:

(1) The four main tasks have different preferences for the architecture, and their correlation coefficients are between -0.003 (segmentation and video recognition) and 0.639 (segmentation and classification). This may be because segmentation can be seen as per-pixel image classification.

(2) Different settings in the same task also have different preferences for the architecture, and their correlation coefficients are between 0.3 (Cls-50-1000 and Cls-50-100) and 0.818 (Cls-50-100 and Cls-10c). From Fig. 1 (a) we also see that different training epochs result in a significant performance change. In this way, the proxy 10-epoch training setting (Radosavovic et al., 2020) may not generalize well to real-world scenarios.

Table 3: Comparative results (%) of NCP on the Cityscapes validation set. The first two architectures are searched using neural predictors that are trained on the Cityscapes dataset with an input size of 512×1024 (Seg) and the resized Cityscapes dataset with an input size of 128×256 (Seg-4x), respectively. The last model is searched by joint optimization of both the two predictors. λ is set to 0.5 during the network codes propagation process. All models are trained from scratch. FLOPs is measured using 512×1024.

Method	Params	FLOPs	Cityscapes 512×1024			Cityscapes 128×512			ADE20K 512×512		
			mIoU	mACC	aACC	mIoU	mACC	aACC	mIoU	mACC	aACC
ResNet18-PSP (Zhao et al., 2017)	12.94M	108.53G	73.21	79.86	95.51	62.66	70.17	93.25	33.45	40.95	77.78
ResNet34-PSP (Zhao et al., 2017)	23.05M	193.12G	76.17	82.66	95.99	64.17	72.37	93.48	32.78	41.23	77.52
HRNet-W18s (Wang et al., 2020)	5.48M	22.45G	76.21	84.43	95.86	64.49	73.58	93.71	34.39	44.65	77.60
HRNet-W18 (Wang et al., 2020)	9.64M	37.01G	**77.73**	**85.61**	96.20	65.19	74.85	93.66	34.84	45.27	77.80
NCP-Net-512 $\times$ 1024 (Fig. 15)	7.91M	29.34G	77.15	84.73	96.01	62.40	71.70	93.34	33.32	42.97	77.41
NCP-Net-128 $\times$ 512 (Fig. 16)	2.61M	31.37G	70.91	80.20	95.21	**65.89**	**75.19**	93.79	27.52	35.88	73.85
NCP-Net-Both (Fig. 17)	7.82M	50.90G	77.35	84.65	**96.23**	64.98	73.82	**93.83**	**35.65**	**45.64**	**77.94**

Table 4: Video recognition results (%) of NCP on the HMDB51 dataset. The first two models are searched using neural predictors that are trained on HMDB51 from scratch (Video) and using models of ImageNet-50-1000 as pretraining (Video-p). The last model is searched by joint optimization of both two predictors. We also show the classification accuracy of the pretrained model on the ImageNet-50-1000 dataset (the column of "Cls-Pretraining"). Note that the last 2D convolutional layer before the final classifier is replaced with a 3D convolutional layer ($3 \times 3 \times 3$) for all models to capture the temporal information. λ is set to 0.5. FLOPs is calculated using an input size of 112×112.

Method	Params	FLOPs	Video-Scratch		Cls-Pretraining		Video-Pretrained	
			top1	top5	top1	top5	top1	top5
ResNet18 (He et al., 2016)	12.10M	0.59G	21.50	54.93	82.72	93.88	27.84	60.50
ResNet34 (He et al., 2016)	22.21M	1.20G	19.89	52.94	83.40	94.28	31.22	63.53
HRNet-W18s (Wang et al., 2020)	14.35M	0.86G	24.55	55.87	83.04	**95.16**	30.96	61.21
HRNet-W18 (Wang et al., 2020)	20.05M	1.41G	24.11	55.33	83.48	94.04	32.48	64.30
NCP-Net-Scratch (Fig. 18)	2.56M	0.69G	**28.47**	62.18	81.88	93.84	37.54	68.50
NCP-Net-Pretrained (Fig. 19)	1.72M	1.00G	27.49	62.01	82.48	94.80	39.14	69.66
NCP-Net-Both (Fig. 20)	2.39M	1.16G	27.14	**62.28**	**83.92**	94.76	**40.21**	**70.29**

(3) The correlation coefficient matrix is related to the model performance of multi-task joint optimization using NP. The higher the correlation coefficient of the two tasks, the greater the gain of joint optimization. If the correlation coefficient of the two tasks is too low, joint optimization may reduce the performance on both tasks.

(4) We show the architecture ranking under different metrics for different tasks in Fig. 1 (b) and (c). We noticed that the three metrics (*i.e.*, mIoU, mAcc, aAcc) in segmentation are positively correlated, which means only one (*e.g.*, mIoU) needs to be optimized to obtain a good model under all three metrics. However, in the 3D object detection task, different metrics are not necessarily related, which makes multi-objective optimization especially important.

B Intra-task Generalizability

In this section, we explore the effect of multiple proxy settings such as different input sizes (512×1024 and 128×256), and pretraining strategies (pretraining from classification or training from the scratch for video recognition) in designing network architectures. We search architectures under single (*e.g.*, 512×1024) or multiple (*e.g.*, both resolutions) to validate the effectiveness of NCP for different intra-task settings.

Tab. 3 and 4 show the intra-task cross-setting generalizability of our searched NCP-Net and some manually-designed networks, such as ResNet (He et al., 2016) and HRNet (Wang et al., 2020), on the Cityscapes dataset and the HMDB51 dataset, respectively. We see that:

(1) Generally, the network searched on a specific setting performs the best under this setting but poor under other settings, which means that NCP can optimize and customize model structures for a specific setting. For example, Tab. 3 shows NCP customizing structures for different input resolutions on Cityscapes, which is essential for practical applications in real-world scenarios.

(2) Benefiting from the high correlation between different settings of the same task, joint optimization of multiple settings within the same task (for both segmentation and video recognition) usually has a positive effect, *i.e.*, improving the performance on each setting, at the cost of the increasing model size (larger FLOPs). However, different tasks may not necessarily correlate, *e.g.*segmentation + video recognition. This can be verified by Spearman's correlation, as discussed in our main paper.

(3) In Tab. 3, we apply the searched segmentation network to the ADE20K dataset (Zhou et al., 2017) to show the generalizability of the searched architectures. The network trained with a large input resolution (512×1024) has better performance than the network trained with a small resolution (128×256).

Moreover, the jointly searched architecture using both resolutions shows better generalizability than those two with a single objective. It also demonstrated that the network searched on our NAS-Bench-MR has stronger transfer capability to real-world scenarios, compared to the previous benchmarks such as (Dong & Yang, 2020; Ying et al., 2019) that uses a small input size.

(4) For both the segmentation and the video recognition tasks, our joint searched networks outperform manually-designed networks, such as HRNet (Wang et al., 2020) and ResNet (He et al., 2016), and achieve better generalizability to other settings and datasets, *e.g.*, ADE20K.

C NCP ON NAS-BENCH-201

In this section, we apply our NCP to the NAS-Bench-201 benchmark (Dong & Yang, 2020) to show its effectiveness. We first briefly introduce NAS-Bench-201 and then state the difference between our NAS-Bench-MR and NAS-Bench-201. Lastly, we detail the experiment.

C.1 NAS-BENCH-201

NAS-Bench-201 employs a DARTS-like (Liu et al., 2019) search space including three stacks of cells, connected by a residual block (He et al., 2016). Each cell is stacked $N = 5$ times, with the number of output channels as 16, 32, and 64 for the first, second, and third stages, respectively. The intermediate residual block is the basic residual block with a stride of 2. There are 6 searching paths in the space of NAS-Bench-201, where each path contains 5 options: (1) zeroize, (2) skip connection, (3) 1-by-1 convolution, (4) 3-by-3 convolution, and (5) 3-by-3 average pooling layer, resulting in 15,625 different models in total.

NAS-Bench-201 train these models on the ImageNet-16 (with an input size of 16×16) and Cifar10 (with an input size of 32×32) datasets. Since the ImageNet dataset is closer to practical applications, we use the models in ImageNet-16 as a comparison in the following section.

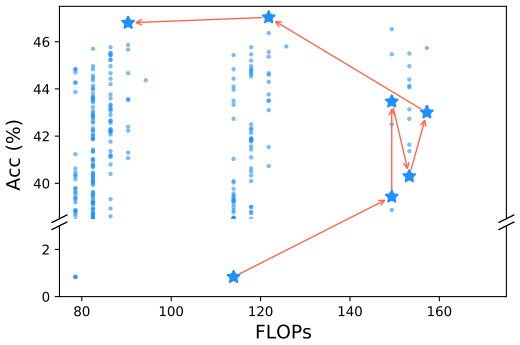


Figure 2: Visualization of the architecture propagation process on the NAS-Bench-201 benchmark (Dong & Yang, 2020) (ImageNet-16). ★ represents the propagated model in each step. NCP finds the optimal structure from a low-Acc and high-FLOPs starting point with only 6 steps by optimizing accuracy and FLOPs constraints simultaneously.

C.2 DIFFERENCES

There are some differences between the NAS-Bench-201 benchmark and our NAS-Bench-MR.

(1) NAS-Bench-201 and NAS-Bench-MR are of different magnitudes (15,625 v.s. 10^{23}). (2) The number of channels/blocks/resolutions is fixed in NAS-Bench-201, while it is searchable in our

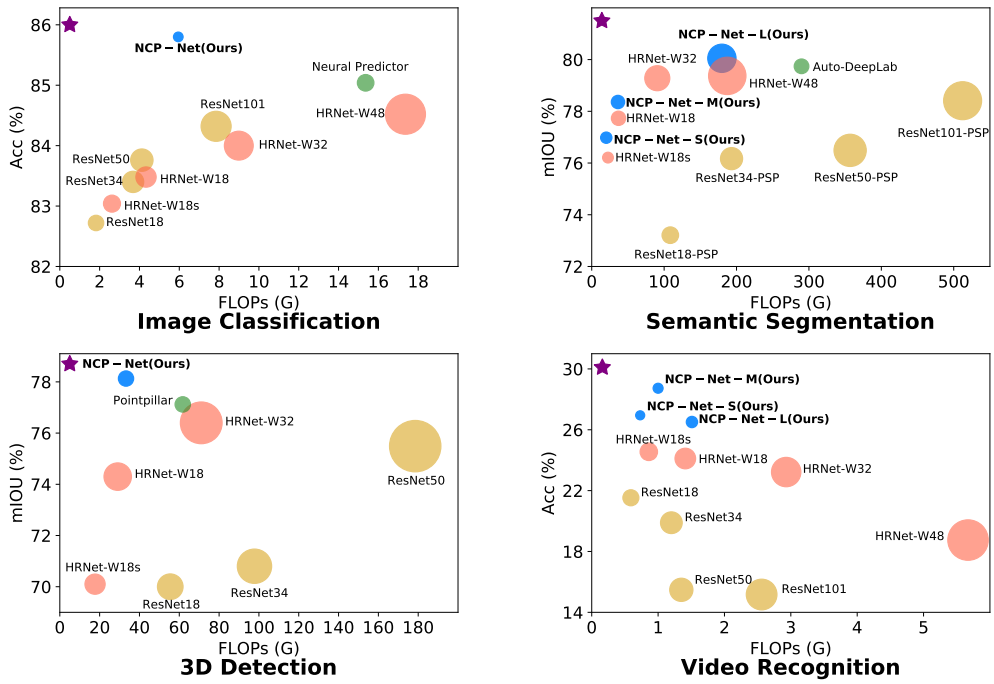


Figure 3: Comparisons of the efficiency (*i.e.*, FLOPs) and the performance (*e.g.*, Acc, mIoU, AP) on four computer vision tasks, *i.e.*, image classification (ImageNet), semantic segmentation (CityScapes), 3D detection (KITTI), and video recognition (HMDB51) between the proposed approach and existing methods. Each method is represented by a circle, whose size represents the number of parameters. ★ represents the optimal model with both high performance and low FLOPs. Our approach achieves superior performance compared to its counterparts on all four tasks.

search space. In this way, our search space is more fine-grained and suitable for customizing to tasks with different preferences (high- and low-level features, deeper and shallower networks, wider and narrower networks, etc.). (3) The architecture in NAS-Bench-201 is represented as a one-hot encoding (choose one of five operators). While in our NAS-Bench-MR, the continuous-valued code is more appealing to the learning of the neural predictor. (4) NAS-Bench-201 is built based on a single setting while NAS-Bench-MR contains multiple settings.

We then conduct experiments and show that despite the above many differences, NCP works well on NAS-Bench-201, demonstrating the generalization ability of NCP to other benchmarks.

C.3 Experiments

In this work, we formulate each architecture in NAS-Bench-201 to a $6*5 = 30$-dimensional one-hot encoding. We then use our continuous network propagation strategy to traverse architectures in the space with higher-Acc and lower-FLOPs as optimization goals. Given an initial one-hot encoding, our NCP treats it as a continuous-valued coding and utilizes the argmax operation to obtain an edited one-hot coding after every several iterations.

In the experiment, we use the argmax operation after every 10 iterations, termed as a step. As shown in Fig. 2, our NCP finds the optimal structure from a low-Acc and high-FLOPs starting point with only 6 steps (60 iterations). The entire search process takes less than 10 seconds. Compared to other neural predictor-based methods such as (Wen et al., 2020; Luo et al., 2020) that need to predict the accuracy of a large number of architectures, NCP is more efficient.

From Fig. 2 we observe that from the second to fourth steps, the performance of the searched model vibrates. This may be because of the instability caused by one-hot encoding. NCP finds an optimal model with an accuracy of 46.8 and FLOPs of only 90.36 (the highest accuracy in NAS-Bench-201 is 47.33%), showing its effectiveness and generalizability.

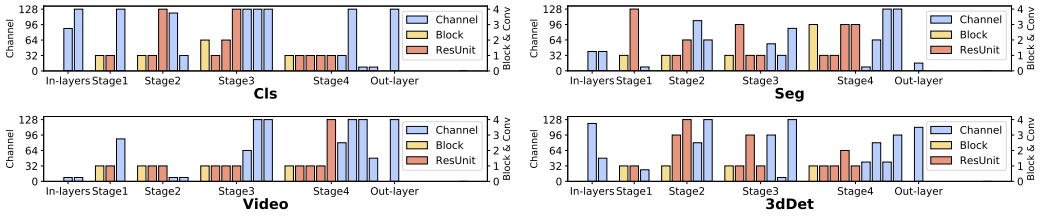


Figure 4: Visualization of the searched models by NCP for four different tasks ($\lambda = 0.5$). The 27-dimensional array in each row represents a network structure.

Algorithm 1 The network propagation process.

1. Learn a neural predictor $\mathcal{F}_W(\cdot)$ and fix it;
2. Initialize an architecture code $\boldsymbol{e}$;
3. Set the target metrics, such as t_{acc} and t_{flops};

for each iteration **do**

4. Re-set new target metrics (optional);
5. Forward $\mathcal{F}_W(\boldsymbol{e})$ and calculate the loss;
6. Back-propagate and calculate the gradient $\Delta\boldsymbol{e}$;
7. Select an optimal dimension l (optional);
8. update $\boldsymbol{e}$ based on the gradient $\Delta\boldsymbol{e}$;

end for

9. Round and decode $\boldsymbol{e}$ to obtain the final architecture.

D NCP ON DIFFERENT SINGLE TASKS

We also use NCP to find the optimal architecture on the four basic vision tasks, *i.e.*, image classification, semantic segmentation, 3D detection, and video recognition. Quantitative comparisons in Fig. 3 show that the architectures found by NCP outperform well-designed networks, such as HRNet (Wang et al., 2020) and ResNet (He et al., 2016).

We visualize the searched architectures by NCP in Fig. 4 to show it can customize different representations for different tasks (objectives). We see that:

(1) Segmentation requires the most high-level information in the last two branches of the last stage, and the video recognition model contains the least low-level semantics in the first two stages.

(2) The classification model contains more channels than other tasks, because the FLOPs constraint for classification is the weakest, *i.e.*, classification costs fewer FLOPs than segmentation under the same architecture.

(3) The 3D detection model mainly utilizes the first two branches, which means it may rely more on high-resolution representations.

E DETAILS OF NCP

In this section, we provide supplementary details and complexity analysis of our NCP. The algorithm of Network Coding Propagation (NCP) is shown in Algorithm 1. Code is available.

E.1 IMPLEMENTATION DETAILS

In this work, for each task, 2000 and 500 structures in the benchmark are used as the training set and validation set to train the neural predictor. The optimization goal is set to higher performance (main evaluation metric of each task) and lower FLOPs.

Intuitively, large models can achieve moderate performance on multiple tasks by stacking redundant structures. On the contrary, the lightweight model tends to pay more attention to task-specific structural design due to limited computing resources, making it more suitable for generalizability evaluation. To this end, we set $\lambda = 0.5$ to obtain lightweight models unless specified.

Table 5: Searching time of predictor-based searching methods. Our NCP is the fastest as it evaluates all dimensions of the code with only one back-propagation without top-K ranking. The time is measured on a Tesla V100 GPU.

Model	Searching Time	Notes
Neural Predictor Wen et al. (2020)	>1 GPU day	Predict the validation metric of 10,000 random architectures and train the top-10 models for final evaluation.
NetAdapt Yang et al. (2018)	5min	Traverse each dimension of the code, predict its per-formance, and then edit the dimension with the highest accuracy improvement.
NCP (Ours)	10s (70 iterations)	Maximize the evaluation metrics along the gradient directions by propagating architectures in the search space.

We employ a three-layer fully connected network with a dropout ratio of 0.5 as the neural predictor. The first layer is a shared layer for all metrics with a dimension of 256. The second layer is a separated layer for each metric with a dimension of 128. Then for each evaluation metric, a 128-by-1 fully-connected layer is used for final regression. During training, we use an Adam optimizer with a weight decay of 1e-7. The initial learning rate is set to 0.01 with batch size 128 on a single GPU for 200 epochs, and decays by the one cycle scheduler. The metric prediction is learned using the smoothL1 loss. In the network propagation process, unless specified, we use continuous propagation with an initial code of $\{b, n = 2; c, i, o = 64\}$ and learning rate of 3 for 70 iterations in all experiments.

E.2 Time Complexity

Tab. 5 shows the searching time of three predictor-based searching methods. NCP is the fastest as it traverses all dimensions of the code with only one back-propagation without the top-K ranking.

Suffering from the random noise in random search and the neural predictor, existing methods (Wen et al., 2020; Luo et al., 2020) often need to train the top-K models for final evaluation, which is costly (*e.g.*, training a segmentation model on Cityscapes costs 7 hours on 8 Tesla V100 GPUs). NCP uses gradient directions as guidance to alleviate the randomness issue.

F Visualization and Analysis

F.1 Analysis of Task Transferring

We visualize the network coding propagation process of our cross-task architecture transferring, *e.g.*, an architecture transferring from the classification task to the segmentation task by using the optimal coding on classification as initialization and using the neural predictor trained on segmentation for optimization. The detailed transferring visualizations of every two of the four tasks are shown in Fig. 5-8 with many interesting findings. For example, all networks in segmentation try to increase the number of channels of the 4th branch of stage 4, while the networks in classification try to decrease it; the networks in video recognition keep it at an intermediate value.

F.2 Analysis of Single- and Multi-task Searching

We visualize the searched architectures (Fig. 9-10) and the network propagation process of each architecture (Fig. 15-20) in Tab. 3-4.

Classification. Manually-designed classification models often use gradually decreasing resolution and gradually increasing the number of channels, because intuitively the learning of classification task requires low-resolution high-level semantic information. However, under different data scales and number of classes, the situation is different, as shown in Fig. 11-14. For example, by reducing the training samples from 1000 to 100 (from Fig. 11 to Fig. 12), the number of convolutional channels and the number of residual units in the 1st branch of stage 4 are increased during propagation, showing the high-resolution low-level information is important when the training samples are insufficient.

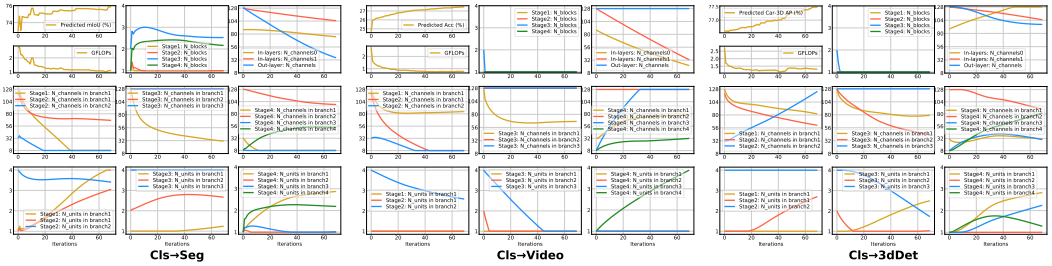


Figure 5: Visualization of our network propagation process of the optimal model in **classification** transferring to other three tasks ($\lambda = 0.5$).

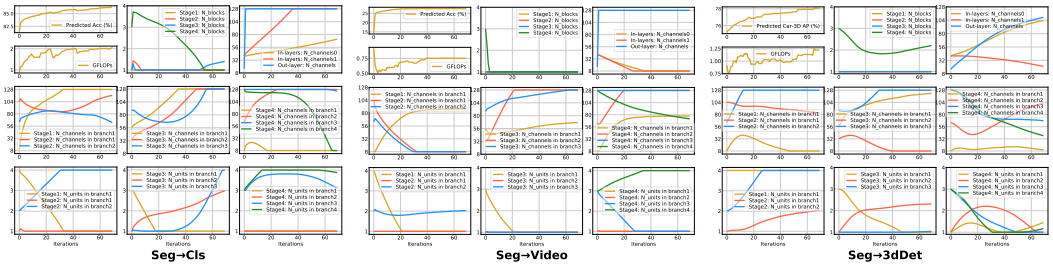


Figure 6: Visualization of our network propagation process of the optimal model in **segmentation** transferring to other three tasks ($\lambda = 0.5$).

Segmentation. From Fig 15, Fig 16, and Fig 17 we observe the differences of the network editing process of segmentation models. For example, compared to the two models optimized at a single resolution, the model optimized at both input resolutions reduces the numbers of residual units of the first two stages and increases the number of blocks of stage 2, resulting in more fusion modules and fewer parallel units.

Segmentation and Video Recognition. From the visualization of found architecture codes in Fig. 9 and 10, we can find, it is not always that the larger the model, the better the performance. Different objectives result in different customized architecture codes. For example, segmentation tends to select architectures with fewer input channels, while video recognition architectures often have more output channels.

REFERENCES

Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Andrew G Berneshawi, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3d object proposals for accurate object class detection. In *NeurIPS*, pp. 424–432, 2015. 2

Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, pp. 3213–3223, 2016. 2

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pp. 248–255. Ieee, 2009. 1

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *ICLR*, 2020. 5

Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*, pp. 3354–3361, 2012. 2

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pp. 770–778, 2016. 4, 5, 7

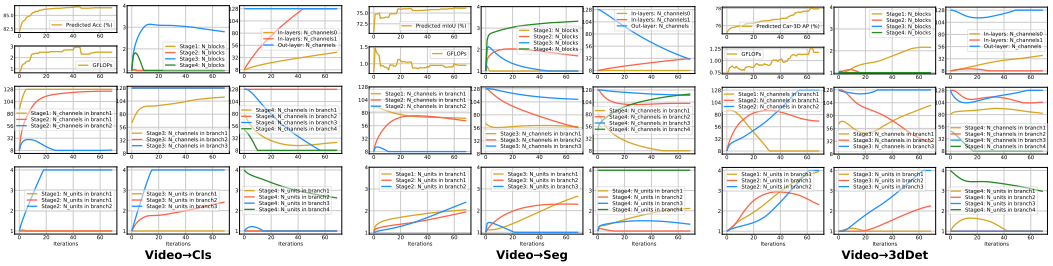


Figure 7: Visualization of our network propagation process of the optimal model in **video action recognition** transferring to other three tasks ($\lambda = 0.5$).

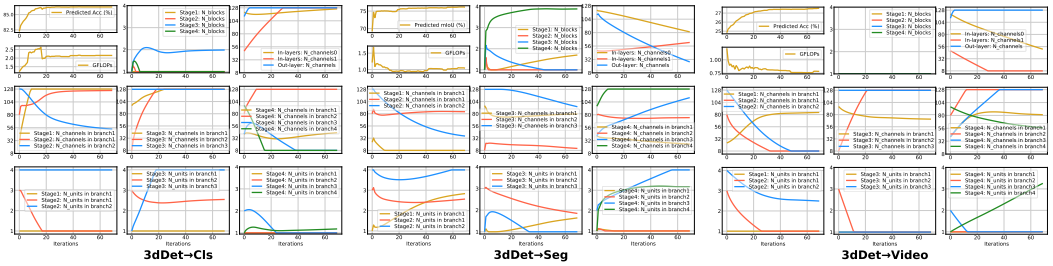


Figure 8: Visualization of our network propagation process of the optimal model in **3D object detection** transferring to other three tasks ($\lambda = 0.5$).

Hildegard Kuehne, Hueihan Jhuang, Estibaliz Garrote, Tomaso Poggio, and Thomas Serre. Hmdb: A large video database for human motion recognition. In *ICCV*, pp. 2556–2563, 2011. 3

Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *CVPR*, pp. 12697–12705, 2019. 2

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. In *ICLR*, 2019. 5

Renqian Luo, Xu Tan, Rui Wang, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture search with gbdt. *arXiv preprint arXiv:2007.04785*, 2020. 6, 8

Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *CVPR*, pp. 10428–10436, 2020. 3

Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, et al. Deep high-resolution representation learning for visual recognition. *IEEE TPAMI*, 2020. 2, 4, 5, 7

Wei Wen, Hanxiao Liu, Yiran Chen, Hai Li, Gabriel Bender, and Pieter-Jan Kindermans. Neural predictor for neural architecture search. In *ECCV*, pp. 660–676. Springer, 2020. 6, 8

Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *ECCV*, pp. 285–300, 2018. 8

Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *ICML*, pp. 7105–7114, 2019. 5

Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *CVPR*, pp. 2881–2890, 2017. 2, 4

Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *CVPR*, pp. 633–641, 2017. 5

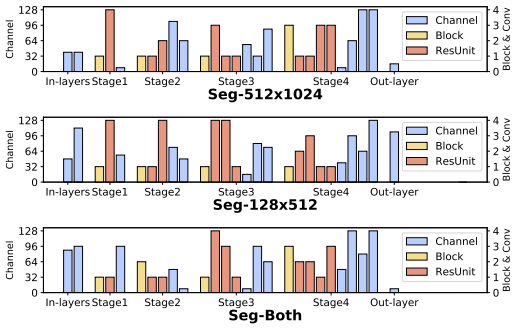


Figure 9: Visualization of the searched **segmentation** models in Tab. 3 by our NCP for intra-task generalizability ($\lambda = 0.5$). The 27-dimensional array in each row represents a network structure.

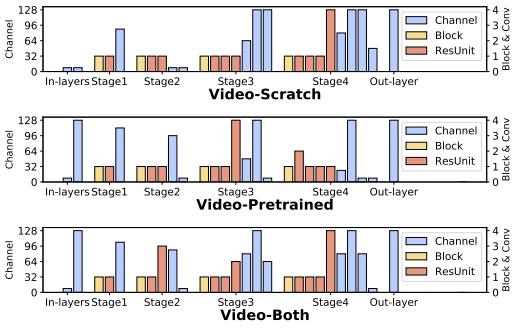


Figure 10: Visualization of the searched **video recognition** models in Tab. 4 by our NCP for intra-task generalizability ($\lambda = 0.5$). The 27-dimensional array in each row represents a network structure.

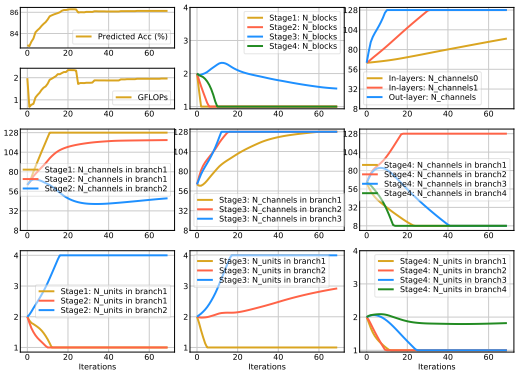


Figure 11: Visualization of our network propagation process of "NCP-Net-A" ($\lambda = 0.7$) for classification.

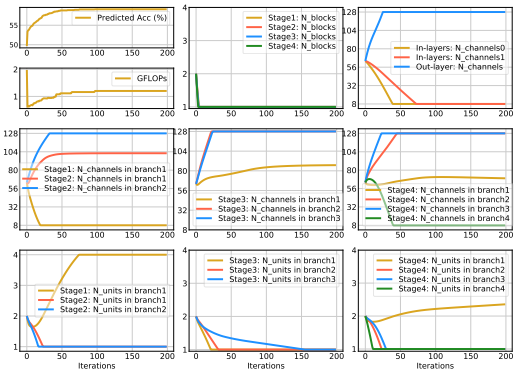


Figure 12: Visualization of our network propagation process of "NCP-Net-B" ($\lambda = 0.7$) for classification.

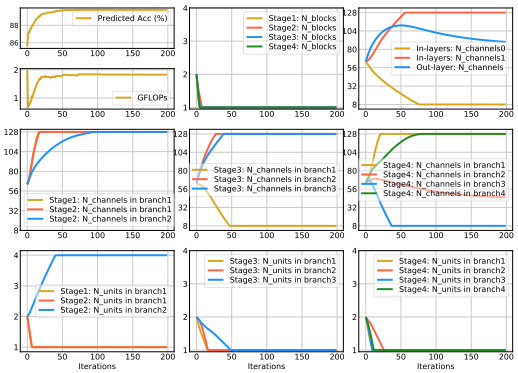


Figure 13: Visualization of our network propagation process of "NCP-Net-C" ($\lambda = 0.7$) for classification.

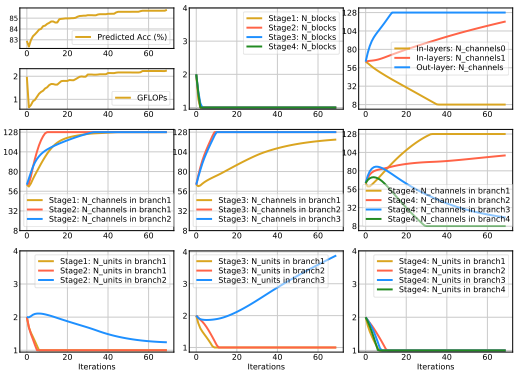


Figure 14: Visualization of our network propagation process of "NCP-Net-ABC" ($\lambda = 0.7$) for classification.

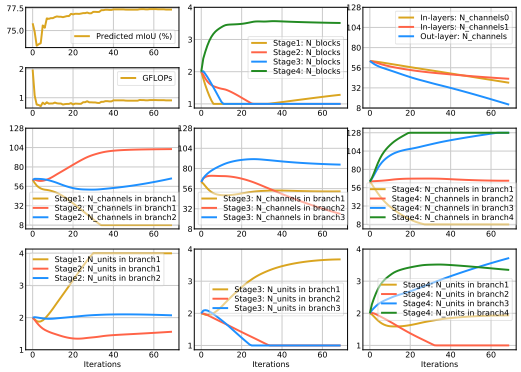


Figure 15: Visualization of our network propagation process of "NCP-Net-512 $\times$ 1024" in Tab. 3 ($\lambda = 0.5$).

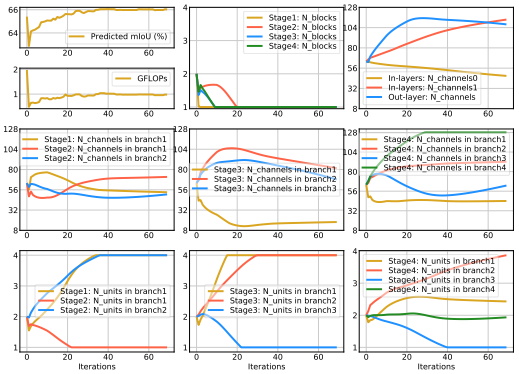


Figure 16: Visualization of our network propagation process of "NCP-Net-128 $\times$ 512" ($\lambda = 0.5$) in Tab. 3 for segmentation.

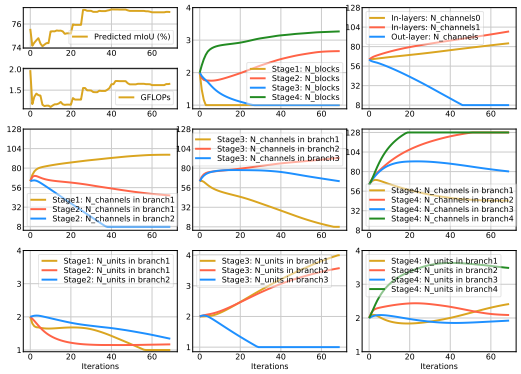


Figure 17: Visualization of our network propagation process of "NCP-Net-Both" ($\lambda = 0.5$) in Tab. 3 for segmentation.

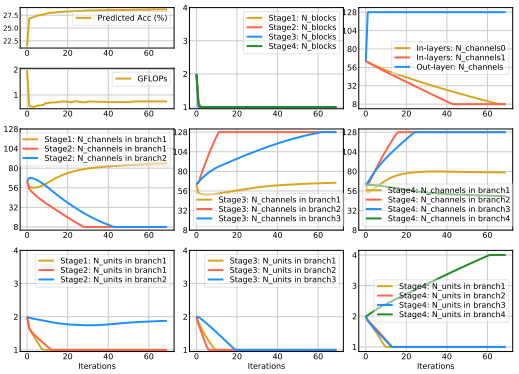


Figure 18: Visualization of our network propagation process of "NCP-Net-Scratch" in Tab. 4 ($\lambda = 0.5$).

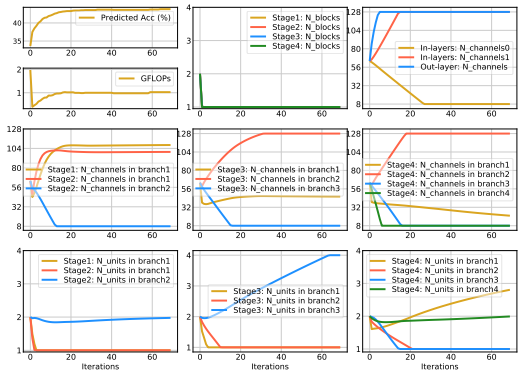


Figure 19: Visualization of our network propagation process of "NCP-Net-Pretrained" ($\lambda = 0.5$) in Tab. 4 for video recognition.

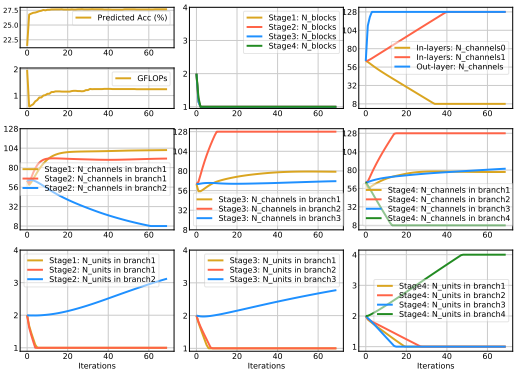


Figure 20: Visualization of our network propagation process of "NCP-Net-Both" ($\lambda = 0.5$) in Tab. 4 for video recognition.