SparseMask: Differentiable Connectivity Learning for Dense Image Prediction

Huikai Wu[*]   Junge Zhang   Kaiqi Huang[†]

Institute of Automation, Chinese Academy of Sciences

University of Chinese Academy of Sciences

{huikai.wu, jgzhang, kaiqi.huang}@nlpr.ia.ac.cn

Abstract

In this paper, we aim at automatically searching an efficient network architecture for dense image prediction. Particularly, we follow the encoder-decoder style and focus on designing a connectivity structure for the decoder. To achieve that, we design a densely connected network with learnable connections, named Fully Dense Network, which contains a large set of possible final connectivity structures. We then employ gradient descent to search the optimal connectivity from the dense connections. The search process is guided by a novel loss function, which pushes the weight of each connection to be binary and the connections to be sparse. The discovered connectivity achieves competitive results on two segmentation datasets, while runs more than three times faster and requires less than half parameters compared to the state-of-the-art methods. An extensive experiment shows that the discovered connectivity is compatible with various backbones and generalizes well to other dense image prediction tasks.

1. Introduction

Dense image prediction is a collection of computer vision tasks that produce a pixel-wise label map for a given image. Such tasks range from low-level vision to high-level vision, including edge detection [40], saliency detection [16] and semantic segmentation [28]. To address these tasks, Long *et al*. propose fully convolutional networks (FCNs), which follow an encoder-decoder style [28]. The encoder is transformed from a pre-trained image classifier, while the decoder combines low-level and high-level features of the encoder to generate the final output. As follows, various methods based on FCNs are proposed, which focus on adjusting the architecture of the decoder manually to achieve a better fusion of multi-level encoder fea-
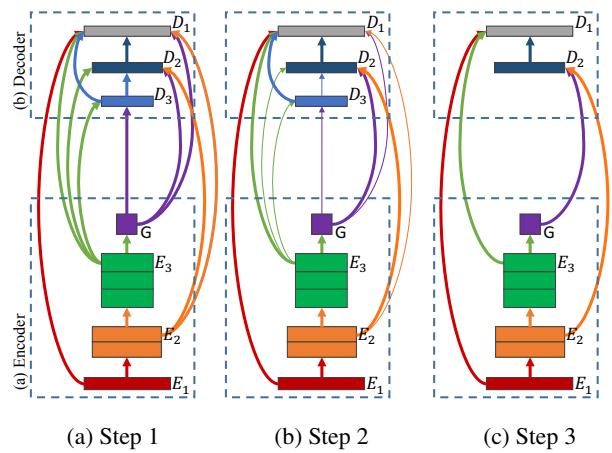

Figure 1: **Framework Overview.** (a) Transform the pre-trained image classifier into a Fully Dense Network with learnable connections. (b) Search the optimal connectivity with the proposed sparse loss in a differentiable manner. (c) Drop the useless connections and stages to obtain the final architecture. Best viewed in color.

tures [40, 32, 20, 16, 12, 41, 44, 30]. However, designing the architecture remains a laborious task, which requires lots of expert knowledge and takes ample time.

Inspired by the success of neural architecture search (NAS) in image classification [49, 50, 24, 26], we aim at automatically designing a decoder for dense image prediction tasks. However, directly employing the methods from image classification is not sufficient, because (1) dense image prediction requires producing a pixel-wise label map, while image classification focuses on predicting a class label, (2) the key to dense image prediction is encoding multi-level features, while image classification aims at extracting global features, and (3) dense image prediction usually requires a pre-trained image classifier for faster training and better performance, while the network for image classification can be designed and trained from scratch. Thus, we face two major challenges: (1) We are required to design

---

a novel search space to combine multi-level features of the pre-trained classifier. (2) The feature maps are usually in high resolution, bringing in heavy computation complexity and memory footprint. Thus, the proposed algorithm needs to design a time and memory efficient network.

To solve the first challenge, we propose a densely connected network named Fully Dense Network (FDN) as the search space, which defines a large set of possible final architectures. FDN follows the encode-decode style, where the encoder is converted from the pre-trained image classifier and the decoder consists of learnable dense connections (Figure 1a). To solve the second challenge, we design a novel loss function to search the optimal connectivity in a differentiable way. The proposed loss forces the weight of each connection to be binary and the connectivity to be sparse (Figure 1b). After training, we prune FDN to obtain the final architecture with sparse connectivity (Figure 1c), which is time and memory efficient. The three steps above (Figure 1) form the proposed method named Sparse-Mask, which can automatically search an efficient decoder for dense image prediction tasks. Particularly, the proposed method focuses on designing the connectivity structure to achieve a better fusion of low-level and high-level features.

To validate the effectiveness of our method, we take a comprehensive ablation study on the Pascal VOC 2012 benchmark [10] with MobileNet-V2 [33] as the backbone. Results show that our method discovers an architecture that outperforms the baseline methods by a large margin within 18 GPU-hours. We then transfer the architecture to other backbones, datasets and tasks. Experiments show that the discovered connectivity has a good generalization ability, which achieves competitive performance, runs more than three times faster, and has less than half parameters.

In summary, we propose a novel method that automatically designs an efficient connectivity structure for the decoder of dense image prediction tasks in a differentiable way. Our contributions are three-folds, (1) we propose Fully Dense Network to define the search space, (2) we introduce a novel loss function to force the dense connections to be sparse, and (3) we conduct comprehensive experiments to validate the effectiveness of our method as well as the generalization ability of the discovered connectivity.

2. Related Work

2.1. Architecture Search

Our work is motivated by differentiable architecture search [34, 37, 26, 1], which is based on the continuous relaxation of the architecture representation, allowing efficient search with gradient descent. [34, 37] propose a grid-like network as the search space, while [26] relax the search space to be continuous and search the space by solving a bilevel optimization problem. Other works in architecture

search employ reinforcement learning [4, 49], evolutionary algorithms [31, 39, 25], and sequential model-based optimization [29, 24] to search the discrete space.

To the best of our knowledge, the most related work is MaskConnect [1], which is designed for image classification. Compared to it, our method is unique in three aspects: (1) We propose a novel sparse loss, which allows an arbitrary number of input features, resulting in a larger and more flexible search space. As for MaskConnect, the number of input features is fixed. (2) We propose an efficient way to concatenate multiple feature maps with different spatial resolutions instead of simply padding and summation. (3) Following the encoder-decoder style [32], we design a search space for dense image prediction tasks, which differs notably from that of MaskConnect.

As for dense image prediction, [34, 37, 11] propose to embed a large number of architectures in a grid-like network. However, the searched network has to be trained from scratch. Differently, our method utilizes a pre-trained image classifier and has a much larger search space. Our work is also complementary to [5], which constructs a recursive search space and employs random search [13] to discover the best architecture. Such a method focuses on extracting multi-scale information from the high-level features, while ours aims at the fusion of the low-level and high-level features. Besides, the search space and search approach are significantly different. AutoDeepLab [23] is also related to our work. However, our method focuses on designing the connectivity of the decoder while AutoDeepLab aims at searching the architecture of the encoder.

2.2. Dense Image Prediction

Currently, there are two prominent paradigms for dense image prediction. [40, 32, 20, 16, 12, 41, 44, 30] propose an encoder-decoder network for combining low-level and high-level features, while [6, 46, 42, 8, 45, 43] utilize dilated convolutions to keep the receptive field size and design a multi-scale context module to process high-level features.

In this paper, we follow the encoder-decoder style, of which the key is designing a connectivity structure to fuse low-level and high-level features. [32] introduce skip connections to combine the decoder features and the corresponding encoder activations, resulting in a network named U-Net. Alternatively, [40] aggregate multiple side outputs to generate the final result. [16] introduce short connections and deep supervision, while [27] enhance the U-Net architecture with an additional bottom-up path. Differently, we aim at automatically designing a sparse connectivity structure to fuse the multi-level features more effectively.

3. Methods

In this paper, we aim at automatically transforming a pre-trained image classifier into an efficient fully convolutional

network (FCN) for dense image prediction tasks. Concretely, given a pre-trained image classifier, we follow the three steps as shown in Figure 1. (1) Transform the classifier into a densely connected network with learnable connections, *i.e.* Fully Dense Network (FDN). See Section 3.1. (2) Employ gradient descent to train the FDN with a novel loss function, which forces the dense connections to be sparse. See Section 3.2. (3) Prune the well-trained FDN to obtain the final network architecture. See Section 3.3.

3.1. Fully Dense Network

We follow the encoder-decoder style [3, 32] in dense image prediction tasks to design the search space, resulting in a super-network named Fully Dense Network (FDN). As shown in Figure 1a, the encoder is a pre-trained image classifier, while the decoder combines multi-level features of the encoder and the decoder with learnable connections.

3.1.1 The Encoder

An image classifier is usually composed of multiple convolution layers, a global average pooling layer [21], and a multi-layer perceptron (MLP). To transform the classifier into an encoder, we simply drop the MLP and keep the rest of the network unchanged. As shown in Figure 1a, the encoder consists of three convolution stages, as well as a global average pooling layer. Each convolution stage contains multiple convolution blocks, such as residual block [15] and inception block [36]. Because the features inside a stage usually have the same spatial and channel dimensions, it's reasonable to assume that the feature of the last block contains the most useful information. We thus restrict the decoder from accessing the other features within a stage. Concretely, the input features of the decoder is limited to the last feature of each encoder stage and the feature after global average pooling, which are noted as E_l and G respectively. l is an index ranging from 1 to L, where L is the number of convolution stages.

3.1.2 The Decoder

We focus on automatically designing the connectivity structure between the encoder and the decoder, as well as the connections inside the decoder. The first problem is to decide the number of stages in the decoder automatically. To achieve that, we initialize the decoder with a large number of stages and employ the search algorithm to select the most important ones. Concretely, the decoder is initialized with the same number of stages as the encoder, *i.e.* L stages. The feature generated by each stage of the decoder is noted as D_l, where l is an index ranging from L to 1, as shown in Figure 1a. Additionally, E_l and D_l have the same spatial dimensions by our design.

The second problem is to automatically choose the input features for each decoder stage. Inspired by DenseNet [17], we propose to initialize the decoder as a densely connected network. Many classic architectures in dense image prediction tasks are a subset of the proposed network, such as U-Net [32]. As shown in Figure 1a, the input features of decoder stage l contains three parts, which are E_i ($i >= l$), D_i ($i > l$) and G. Our method then selects the most important features for each decoder stage automatically. Notably, the proposed FDN is significantly different from DenseNet in three aspects. (1) FDN is densely connected in network-level while DenseNet only has block-level dense connections. (2) The input features of a decoder stage in FDN are from both inside and outside the decoder, following the encoder-decoder style. (3) The input features of a decoder stage in FDN have different spatial dimensions.

The last problem is to efficiently combine the input features inside each decoder stage. In MaskConnect, the input features are padded to the largest spatial/channel dimensions and summed into one feature. The fused feature is then processed by a convolution block. To make the fusion more flexible, we propose to concatenate all the input features channel-wisely and then apply a convolution block to produce the output. However, all the features are required to be up-sampled to the same spatial dimensions before concatenating, which has heavy memory footprint and computation complexity. To reduce memory usage and speed up computation, we show that (1) concatenating the features and then applying convolution is equal to applying convolution to each feature and then take a summation, and (2) the order of bilinear upsampling and point-wise convolution is changeable.[1] Thus, the operations within a decoder stage can be formulated as Equation 1,

$$D_l = \sum_{t \in T_l} w_l^t f_\uparrow(conv_l^t(t)),$$
$$T_l = \{E_i | i >= l\} \cup \{D_i | i > l\} \cup \{G\},$$
(1)

where $f_\uparrow$ is bilinear upsampling, $conv_l^t$ is a convolution block, and w_l^t is a weight to indicate the importance of each connection.

3.2. Searching the Optimal Connectivity

The proposed FDN contains $2^{L(L+1)}$ possible final architectures, which is a huge space to search. For example, there're 2^{30} architectures when the encoder has 5 stages. Our goal is to automatically pick the optimal connectivity out of all the possible ones. To achieve that, we are required to (1) select the most important decoder stages from the L ones, and (2) choose the input features for each selected stage. In practice, the first problem can be reduced to the second one. Concretely, we first select the input features for

---

[1]The proofs are shown in the supplementary material.

all the L decoder stages, and then remove the stages without any in-connections or any out-connections, as shown in Figure 1b and 1c.

Each connection in FDN contains a weight w_l^t to indicate its importance, as shown in Equation 1. To select multiple input features out of all the possible ones for a decoder stage, the most straight forward way is to formulate w_l^t as a binary indicator. If $w_l^t = 1$, the feature t is chosen as the input of the l-th stage. However, directly optimizing over discrete space is data inefficient, which requires lots of computation resources [26]. Alternatively, we propose to relax w_l^t to be a continuous number between 0 and 1, which is then optimized by gradient descent in a differentiable manner.

To relax the discrete optimization problem into a continuous one, w_l^t is required to satisfy a constraint that the value of w_l^t needs to be close to 0 or 1. Besides, $w_l = \{w_l^t | t \in T_l\}$ is desired to be sparse, since we aims at discovering an efficient architecture. To achieve the two constraints, we propose a novel loss function in Equation 2, which forces w_l^t to be binary and w_l to be sparse.

$$L_s(w_l, \alpha) = \mu(L_m(w_l, w_l)) + L_m(\alpha, \mu(w_l)),$$
$$L_m(p, \ q) = -p \times log(q) - (1-p) \times log(1-q), \quad (2)$$

where $\mu(\cdot)$ represents the mean, $\times$ is element-wise multiplication, and α is a hyper-parameter that controls the sparsity ratio. As shown in Figure 2, $L_m(w_l, w_l)$ pushes w_l^t close to either 0 or 1, resulting in a binary-like value. $L_m(\alpha, \mu(w_l))$ forces the mean of w_l to be close to α. When α is close to 0, most values in w_l also tend to be 0, which makes the connections sparse. The final loss is shown in Equation 3,

$$L = L_{task} + \lambda \sum_{l=1}^{L} L_s(w_l, \alpha), \quad (3)$$

where λ controls the balance between the task-oriented loss and the sparse loss.

Notably, w_l^t in Equation 1 cannot entirely indicate the importance of the connection when it is relaxed to be a continuous number. Because the amplitude of $conv_l^t(t)$ also has an influence on the final value of $w_l^t f_\uparrow(conv_l^t(t))$. To reduce the influence of $conv_l^t(t)$, we introduce batch normalization (BN) [18] to normalize the amplitude of $conv_l^t(t)$ into $\mathcal{N}(0, 1)$:

$$D_l = \sum_{t \in T_l} w_l^t f_\uparrow(bn_l^t(conv_l^t(t))). \quad (4)$$

3.3. Pruning Fully Dense Network

To obtain the final architecture, we prune FDN according to the following rules: (1) Drop all connections whose $w_l^t < \sigma$, where σ is a pre-defined threshold. (2) Drop all stages that do not have any input features. (3) Drop all
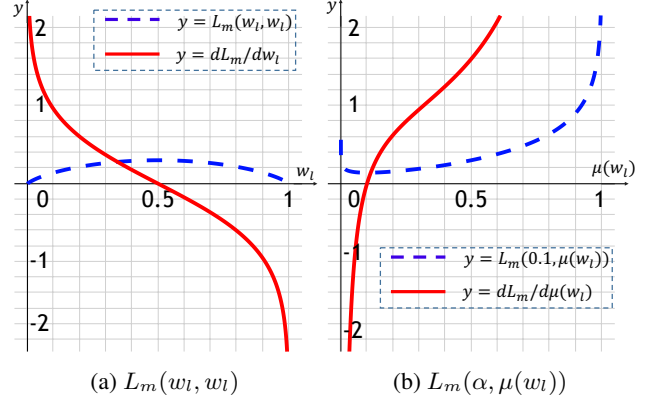

(a) $L_m(w_l, w_l)$ (b) $L_m(\alpha, \mu(w_l))$

Figure 2: **The proposed sparse loss.** The dashed line shows the loss function while the solid line presents the gradient. (a) When minimizing $L_m(w_l, w_l)$, it forces w_l close to either 0 or 1. (b) $L_m(\alpha, \mu(w_l))$ pushes the mean of w_l near α, which makes w_l sparse when α is close to 0.

stages of which the generated feature is not used by other stages. After pruning, w_l^t and bn_l^t in Equation 4 are removed. The network is trained on the target dataset under the supervision of L_{task} only to obtain the final result.

4. Experiments

In this section, we first describe the experimental details of employing our method SparseMask to automatically design a decoder for dense image prediction tasks given a pre-trained image classifier. We then take an extensive experiment to evaluate our approach and compare the discovered architecture with other baseline methods.

4.1. Experimental Setup

The experiment is designed to automatically search a decoder for semantic segmentation with the proposed method. Concretely, we employ our method on the PASCAL VOC 2012 benchmark [10], which contains 20 foreground object classes and one background class. This dataset contains $1, 464$, $1, 449$ and $1, 456$ pixel-wise labeled images for training (*train*), validation (*val*), and testing (*test*) respectively. We augment the *train* set with the extra annotations provided by [14], resulting in $10, 582$ training images (*trainaug*). In our experiment, the *trainaug* set is used for training, while the *val* set is used for testing.

To speed up the search process and reduce memory footprint, we employ MobileNet-V2 [33] as the pre-trained image classifier, which contains 9 convolution stages, resulting in 2^{90} possible architectures[2]. The sparsity ratio α in

---

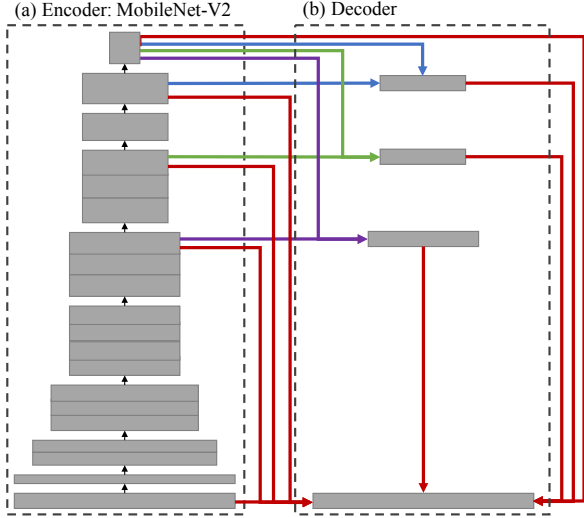[2]FDN based on MobileNet-V2 is shown in the supplementary material.

Figure 3: **The automatically designed architecture.** We employ SparseMask to search an efficient decoder for semantic segmentation with MobileNet-V2 [33] as the encoder. The training dataset is the PASCAL VOC 2012 benchmark [10]. Best viewed in color.

Equation 2 is set as follows,

$$\alpha_l = \min(\frac{2}{|T_l|}, 0.5), l \in [1, L], \tag{5}$$

where $|T_l|$ represents the number of input features for stage l. Under this setting, each decoder stage tends to select two input features. λ in Equation 3 is set to 0.01. The task-related loss L_{task} is set to the pixel-wise cross entropy.

As for training, we follow the protocol presented in [8]. Concretely, we set the learning rates to 0.005 (encoder) and 0.05 (decoder) initially, which decrease to 0 gradually according to the "poly" strategy. For data augmentation, we randomly scale (from 0.5 to 2.0) and left-right flip the input images, which are then cropped to 513×513 and grouped with batch size 16. We train the network for 50 epochs with SGD, of which the momentum is set to 0.9 and the weight decay is set to 4e-5. Notably, the searching is finished within 18 hours on a single Nvidia P100 GPU with 16G memory.

After training, we prune the network with $\sigma = 0.001$ and train the final architecture following the same protocol. The automatically designed architecture is shown in Figure 3. All decoder stages take the feature after global average pooling as the input, which shows its importance for semantic segmentation. The high-level features and the low-level features are also very important, which provides the semantic information and pixel-wise location information respectively. The middle-level information is less useful compared to other features.

Method	mIoU	#Params[3]	FPS[4]
FCN [28]	63.80%	2.22+0.03M	156
Deeplab-V3 [8]	72.51%[5]	2.22+0.67M	50
U-Net [32]	64.72%	2.22+0.16M	97
SparseACN [48]	72.23%	2.22+0.72M	74
FDN (L_{task})	72.72%	2.22+1.93M	37
$L_{task} + L_{sp}$	72.04%	2.22+1.92M	42
$L_{task} + L_{bp}$	73.23%	2.22+0.89M	76
Ours	73.18%	2.22+0.56M	98

Table 1: Performance on Pascal VOC 2012 *val* set with MobileNet-V2 as the backbone.

4.2. Experimental Results

Performance Evaluation   The discovered architecture is evaluated on the *val* set with mean intersection-over-union (mIoU) as the metric. As shown in Table 1, our architecture outperforms the strong baseline FCN [28] and the state-of-the-art method Deeplab-V3 [8] by a large margin. Besides, our method runs much faster than Deeplab-V3 and has fewer parameters. Notably, all the methods employ MobileNet-V2 as the backbone and follow the same training and testing protocol. Besides, no multi-scale testing and left-right flipping are applied to the test images.

Manually Designed Models   Our search space contains many classic architectures designed by experts. To show the architecture discovered by our method is better than the others in the search space, we select two well-known architectures to compare with, namely U-Net [32] and SparseACN [48]. U-Net is a classic architecture that follows the encoder-decoder style. Compared to it, the connectivity pattern of our model is more expressive, which combines more than two input features within a decoder stage and have a better fusion of multi-scale information. As a result, our method outperforms U-Net by a large margin (Table 1). SparseACN proposes a pre-defined sparse connection pattern for densely connected networks, which shows a significant parameter and speed advantages without performance loss. Compared to the pre-defined connection pattern, the connectivity of our model is more flexible and sparser. Experiments in Table 1 show that the discovered model outperforms SparseACN in mIOU, the number of parameters and FPS, which demonstrates the advantage of our approach. We also compare with the super-network, FDN. Results show that our method achieves a superior performance with a much sparser connectivity (Table 1).

---

[3]Parameters come from two parts: the encoder and the decoder.
[4]In this paper, FPS (test phase) is measured on a Nvidia Titan-Xp GPU with a 512×512 image as input.
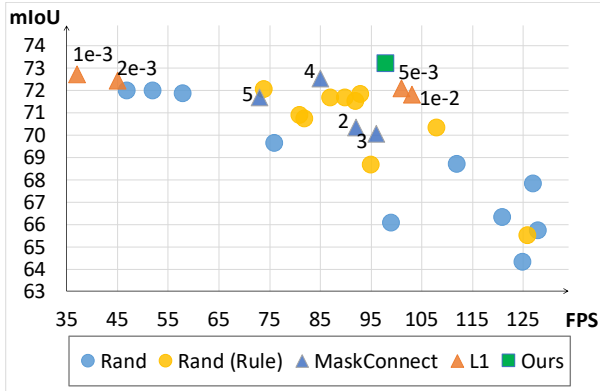[5]w/o COCO pre-training, multi-scale evaluation and deep supervision.

Figure 4: Comparison with random search, MaskConnect [1], and L_1 loss. Best viewed in color.
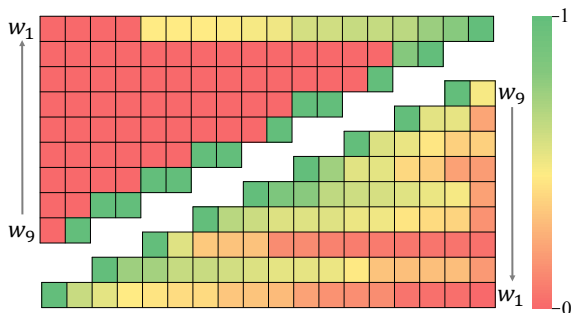

Figure 5: **The Sparsity of Connectivity.** Top-left is the weights trained with the proposed loss, while bottom-right is that with L_1 loss. The proposed loss forces w_l to be binary and the connectivity to be sparse. Best viewed in color.

Randomly Sampled Models We compare with random search to show the effectiveness of the proposed search method. As shown in Figure 4, both "Rand" and "Rand (Rule)" represent the models randomly selected from our search space. However, the models in "Rand (Rule)" follow a constraint observed from the connectivity pattern in Figure 3: the input features of all but the last decoder stages are limited to $\{E_l, G\}$. Results show that (1) Both "Rand" and "Rand (Rule)" contain networks achieving good performance, which shows the effectiveness of our search space. (2) Most networks in "Rand (Rule)" are better than that in "Rand", which shows that our search method can discover a good strategy for connectivity. (3) Our model outperforms all the randomly sampled networks, which shows the effectiveness of the proposed search method. As for the search time, our approach is much faster than random search. Since we only need to train the entire network once, while random search usually needs to train hundreds of subnetworks before finding a good model.

The Sparse Loss To show the effect of the proposed sparse loss, we take an ablation study on the binary part and the sparse part of L_s, noted as L_{bp} and L_{sp}. As shown in Table 1, (1) With L_{task} only or $L_{task} + L_{sp}$, the searched connectivity is dense, resulting in low FPS and large model size. (2) With $L_{task} + L_{bp}$, we can obtain a model with the best mIoU and a reasonable FPS, which shows the effectiveness of the binary part. (3) By adding L_{sp}, the searched model (Ours) achieves similar mIoU with fewer parameters, compared to that of $L_{task} + L_{bp}$.

We also take an experiment to compare with MaskConnect [1] and the widely used L_1 loss. As shown in Figure 4, we employ the search method of MaskConnect on our search space with different hyper-parameters (k), where k is the number of selected input features for each decoder stage. By increasing k from 2 to 5, the searched architecture achieves a better mIoU. However, the architecture of our method outperforms all the models discovered by MaskConnect in both mIoU and FPS. We attribute it to the flexibility of the proposed sparse loss, which enables a much larger search space. Concretely, our method allows an arbitrary number of input features for a decoder stage, while MaskConnect allows exactly k features. Thus, our method can discover a more expressive connection pattern in a search space with fewer constraints.

The architecture designed by L_1 loss also performs worse than ours, as shown in Figure 4. Besides, it runs nearly 3 times slower when the strength is 1e-3, which shows that our loss function is more effective at learning sparse connectivity. By increasing the strength of L_1 loss from 1e-3 to 1e-2, the FPS improves a lot, but the performance drops significantly. The reason is that L_1 loss tends to shrink all the weights, leading to the drop of useful connections after pruning. Concretely, when the strength is 5e-3, the largest dropped connection (6.5e-4) is only 2.6× smaller than the smallest reserved one (1.7e-3), while ours is 200× smaller (1.3e-4 *vs.* 2.7e-2). Thus, the influence of the dropped connections in our method can be ignored. Such results show that our sparse loss is good at obtaining binary values.

To further show the effectiveness of our loss function, we visualize the weight of each connection. As shown in Figure 5, w_l learned with our method is shown in the top-left corner, of which most squares are close to red or green. This indicates that our method is good at approximating binary values. Besides, the proposed loss is also helpful to sparsity, as most squares are in red. The bottom-right corner presents the weights trained with L_1 loss. The color of most squares is between red and green, which means that L_1 loss has little effect to approximate binary numbers. Besides, there's only a few squares in red, which shows that L_1 loss has a marginal effect to sparsity.
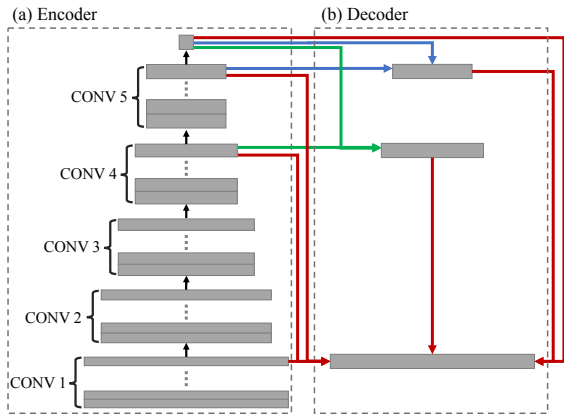
Figure 6: Transfer the automatically designed architecture to other backbones. The features between MobileNet-V2 and the new backbone are matched by spatial dimensions.

Method	mIoU	#Params	FPS	Memory[6]
Deeplab-V2 [7]	79.7%	-	-	-
RefineNet [20]	84.2%	-	-	-
ResNet38 [38]	84.9%	-	-	-
PSPNet [45]	85.4%	45+23M	11.5	0.9+2.3G
DeepLab-V3 [8]	85.7%	45+16M	10.5	0.6+2.3G
EncNet [43]	85.9%	45+18M	11.7	0.8+2.3G
Exfuse [44]	**86.2%**	-	-	-
Ours (Res101)[7]	85.4%	**45+ 7M**	**39.2**	**0.6+1.3G**

Table 2: Performance on the *test* set of PASCAL VOC 2012 benchmark (pre-trained on MS COCO).

Batch Normalization In Equation 4, we employ a BN layer to normalize the feature before multiplying with w_l^t. To verify its effectiveness, we conduct an experiment by removing the introduced BN layers. As a result, the mIoU of the discovered architecture (72.21%) becomes much lower than that with BN layers (73.18%).

5. Transfer to X

The connectivity is searched for semantic segmentation on Pascal VOC 2012 benchmark with MobileNet-V2 as the backbone. To show the generalization ability of the connectivity, we directly employ it to other backbones, datasets, and tasks without tuning.[8]

---

[6]Memory (parameters+features, training phase) are measured on a Nvidia Titan-Xp GPU with a 512×512 image as the input.

[7]http://host.robots.ox.ac.uk:8080/anonymous/WDAEVT.html

[8]The qualitative results are shown in the supplementary material.

5.1. Transfer to Other Backbones

Our method automatically designed a connectivity for the decoder with MobileNet-V2 as the encoder. To show the generalization ability, we transfer the discovered connectivity to other image classifiers (backbones), such as VGG nets [35] and ResNets [15]. However, there're no direct correspondences between the features of different backbones. To transfer the sparse connectivity structure, we propose to match the features by the spatial dimensions. If a match fails, we simply drop the corresponding decoder stage. By following the simple rule, we transfer the automatically designed connectivity to other image classifiers. As shown in Figure 6, the encoder contains 5 convolution stages, in which a down-sampling operation is employed followed by multiple convolution blocks. Then a global average pooling layer is used to extract global information. Such an encoder represents many widely used CNNs including VGG nets and ResNets. Compared to Figure 3, the connectivity structure is similar except that one stage is removed from the decoder because there is no corresponding feature.

To evaluate the performance of the transferred architecture, we conduct an experiment on the PASCAL VOC 2012 benchmark. The training and testing protocol is a little different from Section 4.1. Concretely, we have three steps for training, (1) train the network on MS COCO dataset [22] with learning rate 0.01 for 30 epochs, (2) train the network on the *trainval* set of [14] with learning rate 0.001 for 50 epochs, and (3) train the network on the *trainval* set of the original VOC dataset with learning rate 0.0001 for 50 epochs. The learning rate for the decoder in each step is 10 times larger than the above learning rate. After training, we evaluate the model on the *test* set of PASCAL VOC 2012 benchmark with multi-scale inputs and left-right flipping.

Results in Table 2 show that our method achieves competitive performance, although the architecture is searched on MobileNet-V2 rather than ResNet101. Moreover, our decoder requires half the parameters and runs more than 3 times faster. When training the network, our method occupies much less GPU memory. Concretely, our method can be trained on a Nvidia Titan-Xp GPU with batch size 8, while other methods like EncNet are limited to 4 images.

5.2. Transfer to Other Datasets

The architecture is optimized on the Pascal VOC 2012 benchmark. We employ it on another semantic segmentation dataset ADE20K [47] to verify its generalization ability. ADE20K is a scene parsing benchmark, which contains 150 categories. The dataset includes 20K/2K/3K images for training (*train*), validation (*val*) and testing (*test*).

We train our network on the *train* set for 120 epochs with learning rate 0.01. We then evaluate the model on the *val* set and report the pixel-wise accuracy (PixAcc) and mIoU in Table 3. Our method achieves comparable results to the

Method	PixAcc	mIoU	Score
FCN [28]	71.32%	29.39%	50.36%
SegNet [3]	71.00%	21.64%	46.32%
CascadeNet [47]	74.52%	34.90%	54.71%
RefineNet [20]	-	40.70%	-
PSPNet [45]	81.39%	43.29%	62.34%
EncNet [43]	81.69%	44.65%	63.17%
Ours (Res101)	80.91%	43.47%	62.19%

Table 3: Performance on the *val* set of ADE20K.

Name	PixAcc	mIoU	Score
baseline-DilatedNet	65.41%	25.92%	45.67%
rainbowsecret	71.16%	33.95%	52.56%
WinterIsComing	-	-	55.44%
CASIA_IVA_JD	-	-	55.47%
EncNet-101 [43]	73.74%	38.17%	55.96%
Ours (Res101)	72.99%	38.15%	55.57%

Table 4: Performance on the *test* set of ADE20K.

state-of-the-art PSPNet and EncNet, while requires much fewer parameters and runs much faster. We then fine-tune our network on the *trainval* set for another 20 epochs with learning rate 0.001. The outputs on the *test* set are submitted to the evaluation server. As shown in Table 4, our method outperforms the baseline by a large margin and achieves competitive results compared to EncNet.

5.3. Transfer to Other Tasks

We transfer the network designed for semantic segmentation to other dense image prediction tasks, namely saliency detection [16] and edge detection [40].

Saliency Detection MSRA-B [19] is a widely used dataset for saliency detection, which contains $5,000$ images with a large variation. There are $2,500/500/2,000$ images used for training (*train*), validation (*val*) and testing (*test*) respectively. After training on the *train* set, we evaluate our method on the *test* set. The performance is reported in Table 5. Our method is significantly better than the FCN baseline. Even compared to the state-of-the-art DSS [16], our method achieves comparable result with only $\frac{1}{5}$ parameters. Besides, our method runs nearly 3 times faster.

Edge Detection BSDS500 [2] contains 200 training (*train*), 100 validation (*val*) and 200 test (*test*) images, which is a widely used dataset in edge detection. The *trainval* set is used for training, which is augmented in the same

Method	F_β	MAE	#Params	FPS
FCN [28]	0.861	0.099	2.22+0.01M	186
DSS [16]	0.906	0.054	2.22+2.71M	38
Ours	0.903	0.055	2.22+0.56M	110

Table 5: Saliency detection results on the *test* set of MSRA-B. All methods are based on MobileNet-V2.

Method	ODS	OIS	AP	R50
HED [40]	**0.775**	0.792	0.826	**0.937**
Ours	**0.775**	**0.794**	**0.833**	0.933

Table 6: Edge detection results on the *test* set of BSDS500. Both methods are based on VGG16.

way as [40]. When evaluating on the *test* set, standard non-maximum suppression (NMS) [9] is applied to thin the detected edges. The results are reported in Table 6, where our method outperforms the baseline method HED in two metrics (OIS and AP) and achieves the same ODS.

6. Conclusion

We presented SparseMask, a novel method that automatically designs an efficient network architecture for dense image prediction in a differentiable way, which follows the encoder-decoder style and focuses on the connectivity structure. Concretely, we transformed an image classifier into Fully Dense Network, which contains a large set of possible final architectures and learnable dense connections. With the supervision of the proposed sparse loss, the weight of each connection is pushed to be binary, resulting in an architecture with sparse connectivity. Experiments show that the resulted architecture achieved competitive results on two semantic segmentation datasets, which requires much fewer parameters and runs more than 3 times faster than the state-of-the-art methods. Besides, the discovered connectivity is compatible with various backbones and generalizes well to many other datasets and dense image prediction tasks. Notably, We focus on connectivity search in this paper. However, the proposed search method can be easily extended to layer search. We plan to combine connectivity search and layer search in our future work.

References

[1] Karim Ahmed and Lorenzo Torresani. Maskconnect: Connectivity learning by gradient descent. In *ECCV*, 2018.

[2] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *TPAMI*, 2011.

[3] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv*, 2015.

[4] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *ICLR*, 2017.

[5] Liang-Chieh Chen, Maxwell D Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jonathon Shlens. Searching for efficient multi-scale architectures for dense image prediction. *arXiv*, 2018.

[6] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *ICLR*, 2015.

[7] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *TPAMI*, 2018.

[8] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv*, 2017.

[9] Piotr Dollár and C Lawrence Zitnick. Fast edge detection using structured forests. *TPAMI*, 2015.

[10] Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *IJCV*, 2015.

[11] Damien Fourure, Rémi Emonet, Elisa Fromont, Damien Muselet, Alain Tremeau, and Christian Wolf. Residual convdeconv grid network for semantic segmentation. In *BMVC*, 2017.

[12] Jun Fu, Jing Liu, Yuhang Wang, and Hanqing Lu. Stacked deconvolutional network for semantic segmentation. *arXiv*, 2017.

[13] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D Sculley. Google vizier: A service for black-box optimization. In *SIGKDD*, 2017.

[14] Bharath Hariharan, Pablo Arbeláez, Lubomir Bourdev, Subhransu Maji, and Jitendra Malik. Semantic contours from inverse detectors. In *ICCV*, 2011.

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

[16] Qibin Hou, Ming-Ming Cheng, Xiaowei Hu, Ali Borji, Zhuowen Tu, and Philip Torr. Deeply supervised salient object detection with short connections. In *CVPR*, 2017.

[17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

[18] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv*, 2015.

[19] Huaizu Jiang, Jingdong Wang, Zejian Yuan, Yang Wu, Nanning Zheng, and Shipeng Li. Salient object detection: A discriminative regional feature integration approach. In *CVPR*, 2013.

[20] Guosheng Lin, Anton Milan, Chunhua Shen, and Ian D Reid. Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. In *CVPR*, 2017.

[21] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv*, 2013.

[22] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[23] 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. *ArXiv*, 2019.

[24] Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*, 2018.

[25] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. In *ICLR*, 2018.

[26] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv*, 2018.

[27] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *CVPR*, 2018.

[28] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[29] Renato Negrinho and Geoff Gordon. Deeparchitect: Automatically designing and training deep architectures. *arXiv*, 2017.

[30] Chao Peng, Xiangyu Zhang, Gang Yu, Guiming Luo, and Jian Sun. Large kernel mattersimprove semantic segmentation by global convolutional network. In *CVPR*, 2017.

[31] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc Le, and Alex Kurakin. Large-scale evolution of image classifiers. In *ICML*, 2017.

[32] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

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

[34] Shreyas Saxena and Jakob Verbeek. Convolutional neural fabrics. In *NIPS*, 2016.

[35] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv*, 2014.

[36] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

[37] Tom Veniat and Ludovic Denoyer. Learning time/memory-efficient deep architectures with budgeted super networks. In *CVPR*, 2018.

[38] Zifeng Wu, Chunhua Shen, and Anton van den Hengel. Wider or deeper: Revisiting the resnet model for visual recognition. *arXiv*, 2016.

[39] Lingxi Xie and Alan L Yuille. Genetic cnn. In *ICCV*, 2017.

[40] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In *ICCV*, 2015.

[41] Changqian Yu, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, and Nong Sang. Learning a discriminative feature network for semantic segmentation. In *CVPR*, 2018.

[42] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *ICLR*, 2016.

[43] Hang Zhang, Kristin Dana, Jianping Shi, Zhongyue Zhang, Xiaogang Wang, Ambrish Tyagi, and Amit Agrawal. Context encoding for semantic segmentation. In *CVPR*, 2018.

[44] Zhenli Zhang, Xiangyu Zhang, Chao Peng, Dazhi Cheng, and Jian Sun. Exfuse: Enhancing feature fusion for semantic segmentation. In *ECCV*, 2018.

[45] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *CVPR*, 2017.

[46] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr. Conditional random fields as recurrent neural networks. In *ICCV*, 2015.

[47] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *CVPR*, 2017.

[48] Ligeng Zhu, Ruizhi Deng, Michael Maire, Zhiwei Deng, Greg Mori, and Ping Tan. Sparsely aggregated convolutional networks. In *ECCV*, 2018.

[49] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

[50] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018.