SparseMask: Differentiable Connectivity Learning for Dense Image Prediction

Huikai Wu,* Junge Zhang, Kaiqi Huang
Institute of Automation, 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 automatically designing a connectivity structure for the decoder. To achieve that, we first 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 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. Code is available at https://github.com/wuhuikai/SparseMask.

1. Introduction

Dense image prediction is a collection of computer vision tasks, which produce a pixel-wise label map for a given image. Such tasks range from low-level vision to high-level vision, including edge detection [39], saliency detection [16] and semantic segmentation [27]. To address these tasks, Long *et al.* propose fully convolutional networks (FCNs), which follow an encoder-decoder style [27]. 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 to achieve a better fusion of multi-level features [39, 31, 20, 16, 12, 40, 43, 29]. However, designing the architecture remains a la-

borious task, which requires lots of expert knowledge and takes ample time.

Inspired by the success of automatically designing network architectures for image classification [48, 49, 23, 25], we propose to extend it into dense image prediction. However, directly porting the methods from image classification is not sufficient because (1) dense image prediction requires to produce a pixel-wise label map, while image classification only needs to predict one label, (2) the key to dense image prediction is encoding multi-level features, while image classification merely needs global features, and (3) dense image prediction usually requires a pre-trained image classifier, while the network for image classification can be designed and trained from scratch. Thus, we face two major challenges: (1) We require to design a novel search space for segmentation to combine multi-level features and pre-trained image classifiers. (2) The feature maps in segmentation are usually in high resolution, bringing in heavy computation complexity and memory footprint. Thus, the proposed algorithm needs to automatically design a network that is time and memory efficient.

To solve the first challenge, we transform a pre-trained image classifier into 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 classifier and the decoder consists of learnable dense connections between different features (Figure 1a). To solve the second challenge, we employ gradient descent to search the optimal connectivity from FDN with a novel loss function. 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. As a result, we propose a novel method named SparseMask, which aims at automatically searching an efficient architecture for dense image prediction. Particularly, the proposed method focuses on designing the connectivity for the decoder to achieve a better fusion of low-level and high-level features.

To the best of our knowledge, the most related work to

---

*Part of the work was conducted while I worked as an intern in Preferred Networks (PFN).

our method is MaskConnect [1], which focuses on image classification. Compared to it, our method is unique in three aspects: (1) We propose a novel sparse loss, which forces the weight of each connection to be binary and the connectivity to be sparse. Such a loss allows a decoder stage to have an arbitrary number of input features. The number can be adjusted adaptively for different decoder stages, resulting in a larger and more flexible search space. As for MaskConnect, the number of inputs is fixed. (2) We propose an efficient way to concatenate multiple feature maps with different spatial resolutions instead of simply padding and summation in MaskConnect. (3) Following the encoder-decoder style [31], we design a search space for dense image prediction tasks, which differs notably from classification in MaskConnect.

To validate the effectiveness of our method, we take a comprehensive ablation study on the Pascal VOC 2012 benchmark [10]. 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 pre-trained image classifiers, datasets and tasks. Experiments show that the discovered connectivity has a good generalization ability, which achieves competitive performance compared to the state-of-the-art methods, while runs more than three times faster with less than half parameters.

In summary, we propose a differentiable connectivity learning method, which automatically designs an efficient architecture for dense image prediction. Our contributions are three-folds, (1) we propose Fully Dense Network to define the search space, which is dedicated for dense image prediction, (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 [33, 36, 25, 1], which is based on the continuous relaxation of the architecture representation, allowing efficient search with gradient descent. [33, 36] propose a grid-like network as the search space, while [25] relax the search space to be continuous and train the network by solving a bilevel optimization problem. Other works in architecture search focus on employing reinforcement learning [4, 48], evolutionary algorithms [30, 38, 24], and sequential model-based optimization [28, 23] to search the discrete space.

Recently, [33, 36, 11] propose to embed a large number of architectures in a grid-like network for semantic segmentation, which have to be trained from scratch. Instead, our

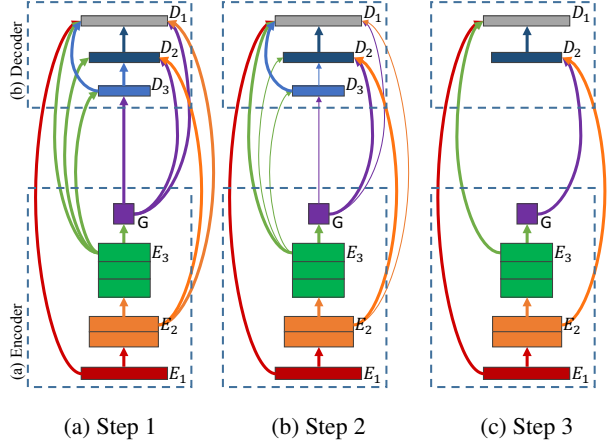(a) Step 1   (b) Step 2   (c) Step 3

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.

method utilizes the pre-trained image classifier and defines a much larger search space of the connectivity. Our work is also complementary to [5], which constructs a recursive search space for dense image prediction 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 low-level and high-level features. Besides, the search space and search approach are significantly different.

2.2. Dense Image Prediction

Currently, there are two prominent directions for dense image prediction. [39, 31, 20, 16, 12, 40, 43, 29] propose an encoder-decoder network for combining low-level and high-level features, while [6, 45, 41, 8, 44, 42] 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 to design a connectivity structure to fuse low-level and high-level features. [31] introduce skip connections to construct U-Net, which combines the encoder features and the corresponding decoder activations. Alternatively, [39] aggregate multiple side outputs to generate the final result. [16] introduce short connections to the skip-layer structures, while [26] enhance the U-Net architecture with an additional bottom-up path. Differently, we aim at automatically designing an efficient connectivity structure with sparse connections.

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. 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, 31] in dense image prediction and design a network named Fully Dense Network (FDN) as the search space. As shown in Figure 1a, the encoder is a pre-trained image classifier, while the decoder combines multi-level features of the encoder with learnable connections.

3.1.1 The Encoder

We take a pre-trained image classifier as the input, which is composed of multiple convolution layers, a global average pooling layer [21], and a multi-layer perceptron (MLP). To form the 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 [35]. Because the features inside a stage usually have the same spatial and channel dimensions, it's reasonable to assume that the feature generated by the last block contains the most information. We thus restrict the decoder from accessing the other features. 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 designing the architecture of the decoder, which includes the connectivity between the encoder and the decoder, as well as the connections inside the decoder. The first problem is to decide the number of decoder stages 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 L stages, which has the same number of stages as the encoder. The feature generated by each

stage 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 to a densely connected network with learnable connections, which contains a large set of possible final architectures. Many classic architectures in dense image prediction is a subset of the proposed network, such as U-net [31]. As shown in Figure 1a, the input features of 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 stage automatically. Notably, the proposed FDN is significantly different from DenseNet in four aspects. (1) FDN is densely connected in network-level while DenseNet only has block-level dense connections. (2) FDN contains input features from both inside and outside the decoder, following the encoder-decoder style. (3) The spatial dimensions of each input feature are different in FDN. (4) FDN contains learnable dense connections, which becomes sparser as the training progresses.

The last problem is to efficiently combine the input features inside each decoder stage. In MaskConnect, the input features are padded to the largest resolution and summed into one feature, which 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 require to be up-sampled to the same spatial dimensions before concatenating, which occupies large memory space and costs many computation resources. 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 architecture can be reformulated as Equation 1,

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

where $f_\uparrow$ is bilinear upsampling, $conv_l^t$ represents a convolution block, and w_l^t is the learnable weight for each connection.

3.2. Learning Sparse Connections

The proposed FDN contains $2^{L(L+1)}$ possible final architectures. Concretely, 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 require to (1) select the most important stages from

---

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

the L ones, and (2) choose the input features for each selected stage, as shown in Section 3.1. In practice, the first problem can be reduced to the second one. Concretely, we first select the input features for all the L stages, and then remove the stages without in-connections or out-connections, as shown in Figure 1b and 1c. Thus, we focus on automatically choosing the input features for each stage.

As shown in Equation 1, each connection contains a learnable weight w_l^t, which is 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 binary numbers is data inefficient, which requires lots of computation resources [25]. 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.

The weight w_l is desired to satisfy two requirements, (1) w_l^t needs to approximate binary numbers, of which the value is around 0 or 1, and (2) w_l requires to be sparse, in which most values are close to 0. To achieve that, we propose a novel loss function, which forces w_l^t to be binary and the connections to be sparse, as shown in Equation 2.

$$
\begin{aligned}
L_s(w_l, \alpha) &= \mu(L_m(w_l, w_l)) + L_m(\alpha, \mu(w_l)), \\
L_m(p,\ q)\ &= -\,p * log(q) - (1-p) * log(1-q),
\end{aligned}
\tag{2}
$$

where $\mu(\cdot)$ represents the mean, $*$ 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,

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

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

Notably, w_l^t itself cannot represent the importance of the input feature 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 remove the influence of $conv_l^t(t)$, we introduce batch normalization (BN) [18] into Equation 1, which normalizes the amplitude of $conv_l^t(t)$ into $\mathcal{N}(0,1)$ as shown in Equation 4:

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

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 predefined threshold. (2) Drop all

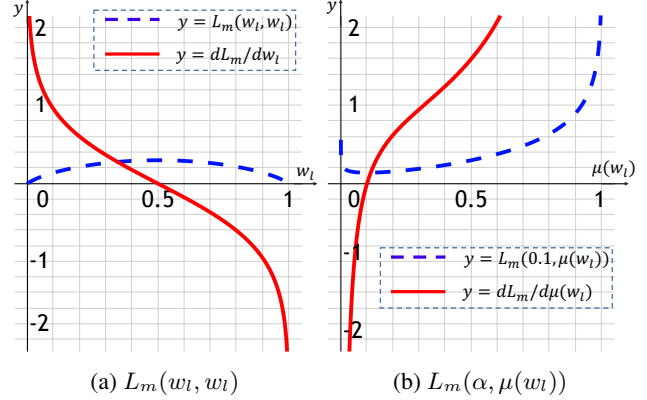(a) $L_m(w_l, w_l)$ (b) $L_m(\alpha, \mu(w_l))$

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

stages that do not have any input features. (3) Drop all stages of which the generated feature is not used by other stages. After pruning, the network is then trained on the target dataset under the supervision of L_{task} only. Notably, w_l^t and bn_l^t in Equation 4 are removed when training the discovered architecture.

4. Experiments

In this section, we first describe the experimental details of automatically designing a network architecture with our method. We then conduct an extensive experiment to evaluate our method and the discovered architecture. To show the generalization ability, we conduct a thorough experiment by transferring the discovered connectivity structure to other pre-trained image classifiers, datasets, and dense image prediction tasks.

4.1. Learning Sparse Connectivity

Experimental Setup The experiment is designed to automatically search a network architecture 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 $1,456$ pixelwise 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.

We employ MobileNet-V2 [32] as the pre-trained image classifier, which contains 9 convolution stages, resulting in

2^{90} possible architectures. The sparsity ratio α in Equation 2 is set as follows,

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

where $|w_l|$ represents the number of elements in w_l. With 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 loss.

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 as the "poly" strategy. For data augmentation, we randomly scale (from 0.5 to 2.0) and left-right flip the input images. The images 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 weight decay is set to 4e-5. Notably, the training 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. The feature after global average pooling plays an essential role in semantic segmentation, as each stage takes it as the input. The high-level features with small spatial sizes and the low-level features with large spatial dimensions 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.

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 [27] and the state-of-the-art method Deeplab-V3 [8] by a large margin. Besides, our method runs much faster than Deeplab-V3 with 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.

4.2. Ablation Study

Models in the Search Space Our search space is huge, which contains various classical encoder-decoder architectures. 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, namely U-Net [31]
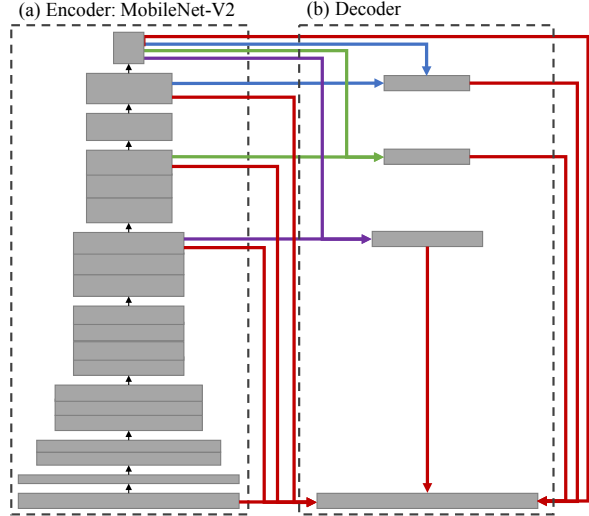
---

[2]Params comes from two parts: the encoder and the decoder.

[3]In this paper, FPS (test phase) is measured on a Nvidia Titan-Xp GPU with a 512×512 image as input.

[4]w/o COCO pre-training, multi-scale evaluation and deep supervision.

[5]After learning with L_1, none of the connections are removed.

Figure 3: **The automatically designed architecture by our method.** We employ MobileNet-V2 [32] as the pretrained image classifier and aim at designing an efficient architecture for segmentation. The training dataset is the PASCAL VOC 2012 benchmark [10]. Best viewed in color.

Method	mIoU	#Params[2]	FPS[3]
FCN [27]	63.80%	2.22+0.03M	156
Deeplab-V3 [8]	72.51%[4]	2.22+0.67M	50
U-net [31]	64.72%	2.22+0.16M	97
SparseACN [47]	72.23%	2.22+0.72M	74
MaskConnect [1]	70.34%	2.22+0.39M	92
Ours w/ L_1 (FDN)[5]	72.72%	2.22+1.93M	37
Ours	**73.18%**	2.22+0.56M	98

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

and SparseACN [47], as shown in Table 1. U-net is a classical architecture that follows the encoder-decoder style. Compare to it, the connectivity pattern of our model is more expressive, which combines more than two input features for a decoder stage and have a better fusion of multi-scale information. As a result, our method outperforms U-net by a large margin. SparseACN proposes a predefined sparse connection pattern for densely connected networks, which shows a significant parameter and speed advantages without performance loss. Compared to the predefined connection pattern, the connectivity of our model is more flexible and sparser. Experiments show that the discovered model outperforms SparseACN in mIOU, the number of parameters, and FPS, which demonstrates the advantage of learnable connections. We also compare with the whole network

FDN without pruning any connections. Results show that our method achieves a superior performance.

The Sparse Loss To show the effect of the proposed sparse loss, we compare it with MaskConnect [1] and the widely used L_1 loss. As shown in Table 1, the architecture discovered by our method outperforms that of MaskConnect a large margin, while the number of parameters and speed are comparable. We attribute it to the flexibility of the proposed sparse loss, which enables the search in a much larger space. Concretely, our method allows an arbitrary number of input features for a decoder stage, while in MaskConnect, each decoder stage contains exactly k input features. Thus, our method can discover more expressive connection patterns in a search space with fewer constraints.

The architecture designed by L_1 loss also performs worse than ours. Moreover, it runs nearly 3 times slower and contains 4 times more parameters. We also notice that the weights learned with L_1 loss are all larger than the threshold 0.001, resulting in 90 connections after pruning. As for our method, there's only 15 connections in the final architecture, which shows that our loss function is more effective at learning sparse connectivity. As a result, our method can discover architectures with higher FPS and fewer parameters. Besides, the proposed sparse loss is also good at obtaining binary values. In our method, the largest weight of the dropped connections (1.342e-4) is about 200× smaller than the smallest weight of the reserved connections (2.666e-2). Since the weights of all the dropped connections are close to 0, the influence of these connections can be ignored.

To further show the effectiveness of our loss function, we visualize the weight of each connection. As shown in Figure 4, 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.

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%), which shows the importance of Equation 4.
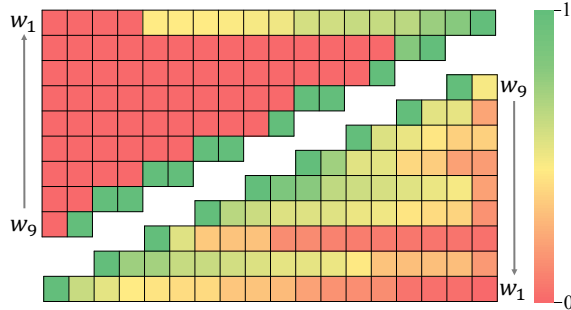
Figure 4: **The Sparsity of Connectivity.** Top-left is the weights trained with the proposed loss, while bottom-right is that with L_1 loss. The proposed loss forces w_l to be close to either 0 or 1 and the connectivity to be sparse. Best viewed in color.
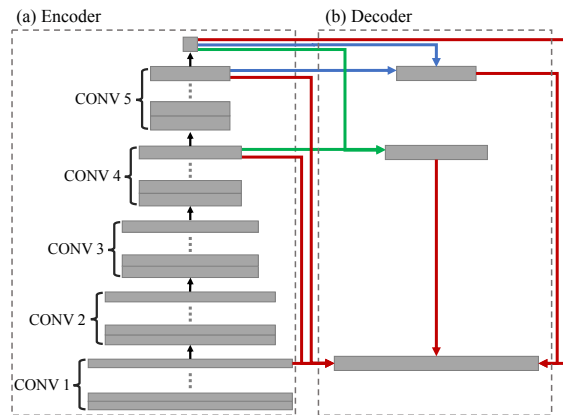

Figure 5: Transfer the automatically designed architecture to other backbones. We match the features between MobileNet-V2 and the new backbone based on spatial dimensions.

4.3. Transfer to Other Backbones

Our method automatically designed a network architecture based on MobileNet-V2. In this section, we transfer the discovered architecture to other image classifiers (also known as backbones), such as VGG nets [34] and ResNets [15]. The key is to transfer the sparse connectivity structure. However, there's no direct correspondences between the features of different backbones. Instead, we propose to match the features according to the spatial dimensions. If a match fails, we simply drop the corresponding stage. By following such rules, we transfer the automatically designed connectivity to other image classifiers. As shown in Figure 5, the encoder contains 5 convolution stages, in which a down-sampling operation is employed followed by multiple convolution blocks. Then a global av-
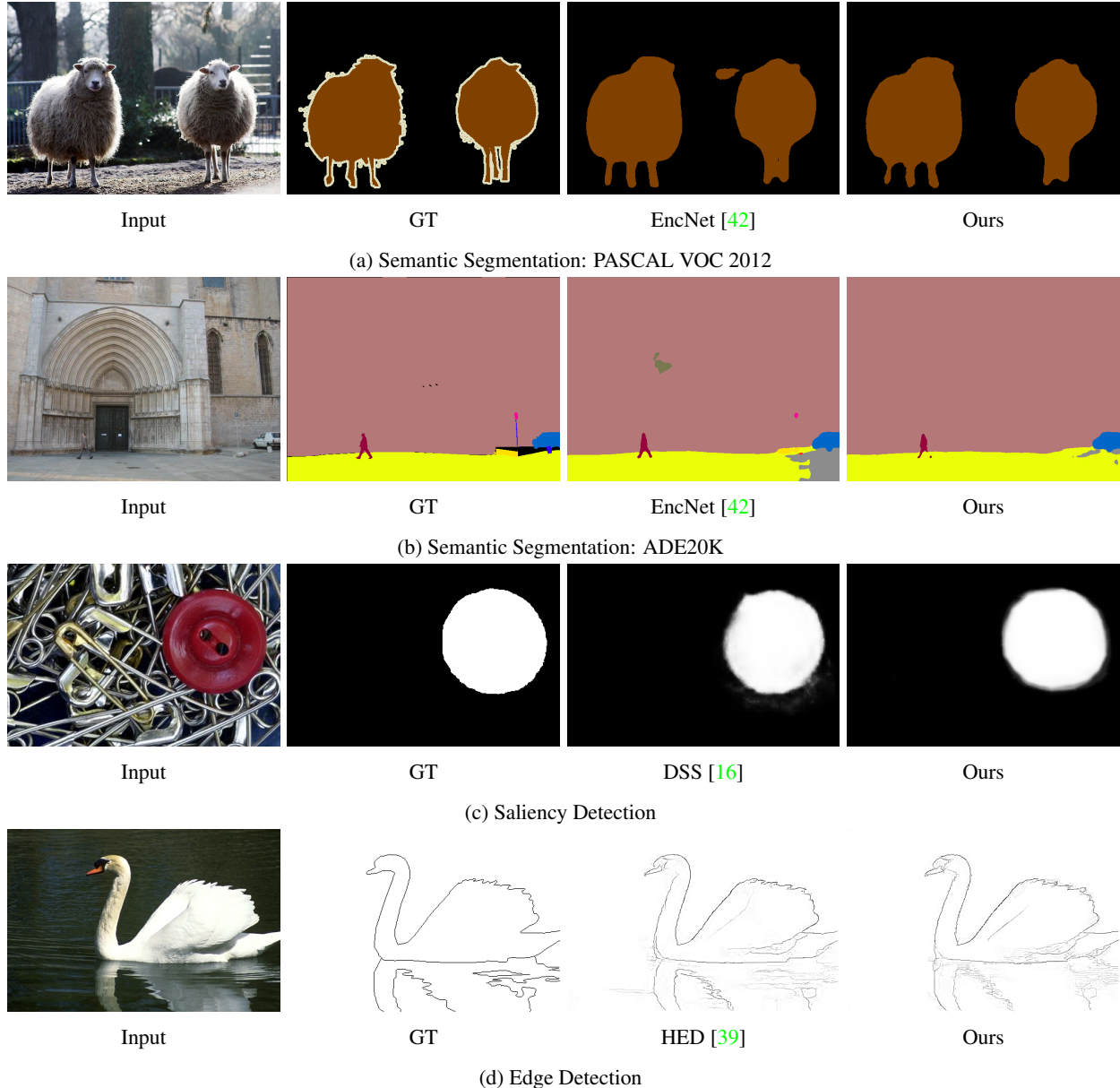
(a) Semantic Segmentation: PASCAL VOC 2012

Input GT EncNet [42] Ours

(b) Semantic Segmentation: ADE20K

Input GT EncNet [42] Ours

(c) Saliency Detection

Input GT DSS [16] Ours

(d) Edge Detection

Input GT HED [39] Ours

Figure 6: **Qualitative Results.** Our method is not only quantitively but also qualitatively comparable to the baseline method.

erage 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.

The results are shown in Table 2. Our method achieves competitive performance in mIoU compared to the state-of-the-art methods, although the architecture is trans-

Method	mIoU	Params	FPS	Memory
Deeplab-V2 [7]	79.7%	-	-	-
RefineNet [20]	84.2%	-	-	-
ResNet38 [37]	84.9%	-	-	-
PSPNet [44]	85.4%	45+23M	11.5	0.9+2.3G
DeepLab-V3 [8]	85.7%	45+16M	10.5	0.6+2.3G
EncNet [42]	85.9%	45+18M	11.7	0.8+2.3G
Exfuse [43]	**86.2%**	-	-	-
Ours (Res101) [6]	85.4%	**45+ 7M**	**39.2**	**0.6+1.3G**

Table 2: Performance on the *test* set of PASCAL VOC 2012 benchmark with pre-training on MS COCO dataset. Memory Usage (parameters+features, training phase) are measured on a Nvidia Titan-Xp GPU with a 512×512 image as the input.

Method	PixAcc%	mIoU%	Score%
FCN [27]	71.32	29.39	50.36
SegNet [3]	71.00	21.64	46.32
CascadeNet [46]	74.52	34.90	54.71
RefineNet [20]	-	40.70	-
PSPNet [44]	81.39	43.29	62.34
EncNet [42]	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 [42]	73.74	38.17	55.96
Ours (Res101)	72.99	38.15	55.57

Table 4: Segmentation result on the *test* set of ADE20K.

Method	F_β	MAE	Params	FPS
FCN [27]	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 without Dense-CRF as post-processing.

ferred from MobileNet-V2 and not designed specifically for ResNet101. Moreover, our decoder requires half the parameters and runs more than 3 times faster. When training the network, our method occupies much fewer 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. The qualitative results on the *val* set of the original VOC benchmark are shown in Figure 6 (a).

4.4. Transfer to Other Datasets

The architecture is optimized on the Pascal VOC 2012 benchmark. We now employ it on another semantic segmentation dataset ADE20K [46] to verify its generalization ability. ADE20K dataset is a scene parsing benchmark, which contains 150 stuff/object 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 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. The qualitative results on the *val* set are shown in Figure 6 (b).

4.5. 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 [39].

4.5.1 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 and $2,000$ 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, our method achieves comparable result with only $\frac{1}{5}$ parameters. Besides, our method runs nearly 3 times faster than DSS. The qualitative results on the *test* set are shown in Figure 6 (c).

4.5.2 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 way as [39]. When

Method	ODS	OIS	AP	R50
HED [39]	**0.775**	0.792	0.826	**0.937**
Ours (VGG16)	**0.775**	**0.794**	**0.833**	0.933

Table 6: Edge detection results on the *test* set of BSDS500.

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. The qualitative results on the *test* set are shown in Figure 6 (d).

5. Conclusion

We presented SparseMask, a novel method that automatically designs an efficient network architecture for dense image prediction in a differentiable manner, which follows the encoder-decoder style and focuses on the connectivity structure. Concretely, we transformed a pre-trained 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 either 0 or 1, resulting in a network 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 sparse connectivity structure is compatible with various backbones and generalizes well to many other datasets and dense image prediction tasks.

References

[1] K. Ahmed and L. Torresani. Maskconnect: Connectivity learning by gradient descent. In *ECCV*, 2018. 2, 5, 6

[2] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *TPAMI*, 2011. 8

[3] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv*, 2015. 3, 8

[4] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. In *ICLR*, 2017. 2

[5] L.-C. Chen, M. D. Collins, Y. Zhu, G. Papandreou, B. Zoph, F. Schroff, H. Adam, and J. Shlens. Searching for efficient multi-scale architectures for dense image prediction. *arXiv*, 2018. 2

[6] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *ICLR*, 2015. 2

[7] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *TPAMI*, 2018. 8

[8] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv*, 2017. 2, 5, 8

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

[10] M. Everingham, S. A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *IJCV*, 2015. 2, 4, 5

[11] D. Fourure, R. Emonet, E. Fromont, D. Muselet, A. Tremeau, and C. Wolf. Residual conv-deconv grid network for semantic segmentation. In *BMVC*, 2017. 2

[12] J. Fu, J. Liu, Y. Wang, and H. Lu. Stacked deconvolutional network for semantic segmentation. *arXiv*, 2017. 1, 2

[13] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley. Google vizier: A service for black-box optimization. In *SIGKDD*, 2017. 2

[14] B. Hariharan, P. Arbeláez, L. Bourdev, S. Maji, and J. Malik. Semantic contours from inverse detectors. In *ICCV*, 2011. 4, 7

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 3, 6

[16] Q. Hou, M.-M. Cheng, X. Hu, A. Borji, Z. Tu, and P. Torr. Deeply supervised salient object detection with short connections. In *CVPR*, 2017. 1, 2, 7, 8

[17] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017. 3

[18] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv*, 2015. 4

[19] H. Jiang, J. Wang, Z. Yuan, Y. Wu, N. Zheng, and S. Li. Salient object detection: A discriminative regional feature integration approach. In *CVPR*, 2013. 8

[20] G. Lin, A. Milan, C. Shen, and I. D. Reid. Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. In *CVPR*, 2017. 1, 2, 8

[21] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv*, 2013. 3

[22] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 7

[23] C. Liu, B. Zoph, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *ECCV*, 2018. 1, 2

[24] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. In *ICLR*, 2018. 2

[25] H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. *arXiv*, 2018. 1, 2, 4

[26] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia. Path aggregation network for instance segmentation. In *CVPR*, 2018. 2

[27] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 1, 5, 8

[28] R. Negrinho and G. Gordon. Deeparchitect: Automatically designing and training deep architectures. *arXiv*, 2017. 2

[29] C. Peng, X. Zhang, G. Yu, G. Luo, and J. Sun. Large kernel mattersimprove semantic segmentation by global convolutional network. In *CVPR*, 2017. 1, 2

[30] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *ICML*, 2017. 2

[31] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015. 1, 2, 3, 5

[32] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018. 4, 5, 12

[33] S. Saxena and J. Verbeek. Convolutional neural fabrics. In *NIPS*, 2016. 2

[34] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv*, 2014. 6

[35] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016. 3

[36] T. Veniat and L. Denoyer. Learning time/memory-efficient deep architectures with budgeted super networks. In *CVPR*, 2018. 2

[37] Z. Wu, C. Shen, and A. v. d. Hengel. Wider or deeper: Revisiting the resnet model for visual recognition. *arXiv*, 2016. 8

[38] L. Xie and A. L. Yuille. Genetic cnn. In *ICCV*, 2017. 2

[39] S. Xie and Z. Tu. Holistically-nested edge detection. In *ICCV*, 2015. 1, 2, 7, 8, 9

[40] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu, and N. Sang. Learning a discriminative feature network for semantic segmentation. In *CVPR*, 2018. 1, 2

[41] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. In *ICLR*, 2016. 2

[42] H. Zhang, K. Dana, J. Shi, Z. Zhang, X. Wang, A. Tyagi, and A. Agrawal. Context encoding for semantic segmentation. In *CVPR*, 2018. 2, 7, 8

[43] Z. Zhang, X. Zhang, C. Peng, D. Cheng, and J. Sun. Exfuse: Enhancing feature fusion for semantic segmentation. In *ECCV*, 2018. 1, 2, 8

[44] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. In *CVPR*, 2017. 2, 8

[45] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. Torr. Conditional random fields as recurrent neural networks. In *ICCV*, 2015. 2

[46] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba. Scene parsing through ade20k dataset. In *CVPR*, 2017. 8

[47] L. Zhu, R. Deng, M. Maire, Z. Deng, G. Mori, and P. Tan. Sparsely aggregated convolutional networks. In *ECCV*, 2018. 5

[48] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. 1, 2

[49] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018. 1

Theorems

We present two theorems in Section 3.1.2, of which the proofs are given in this section.

Theorem 1. *Concatenating the features and then applying convolution is equal to applying convolution to each feature and then take a summation.*

Proof. Given M input features F_{in}^m with shape $N \times C_{in}^m \times H \times W$, the concatenated feature is noted as F_{in} with shape $N \times C_{in} \times H \times W$, where $C_{in} = \sum_{m=0}^{M-1} C_{in}^m$. The corresponding convolution kernel is noted as W with shape $C_{out} \times C_{in} \times KH \times KW$, which can be split into M weights W^m with shape $C_{out} \times C_{in}^m \times KH \times KW$. The output feature F_{out} is represented as following:

$$
\begin{aligned}
F_{out}[n, c_{out}, h, w] &= conv(F_{in}, W)[n, c_{out}, h, w] \\
&= \sum_{kh,kw} \sum_{c_{in}=0}^{C_{in}-1} W[c_{out}, c_{in}, kh, kw] F_{in}[n, c_{in}, h+kh, w+kw] \\
&= \sum_{kh,kw} \sum_{m=0}^{M-1} \sum_{c_{in}=0}^{C_{in}^m-1} W^m[c_{out}, c_{in}, kh, kw] F_{in}^m[n, c_{in}, h+kh, w+kw] \\
&= \sum_{m=0}^{M-1} \sum_{kh,kw} \sum_{c_{in}=0}^{C_{in}^m-1} W^m[c_{out}, c_{in}, kh, kw] F_{in}^m[n, c_{in}, h+kh, w+kw] \\
&= \sum_{m=0}^{M-1} conv(F_{in}^m, W^m)[n, c_{out}, h, w].
\end{aligned}
\tag{6}
$$

$\square$

Theorem 2. *The order of bilinear upsampling and point-wise convolution is changeable.*

Proof. The input feature is F_{in} with shape $N \times C_{in} \times H_{in} \times W_{in}$, while the corresponding convolution kernel is W with shape $C_{out} \times C_{in} \times 1 \times 1$. The output features F_{out} is then represented as following:

$$
\begin{aligned}
F_{out}[n, c_{out}, h_{out}, w_{out}] &= conv(f_\uparrow(F_{in}), W)[n, c_{out}, h_{out}, w_{out}] \\
&= \sum_{c_{in}} W[c_{out}, c_{in}, 0, 0] f_\uparrow(F_{in})[n, c_{in}, h_{out}, w_{out}] \\
&= \sum_{c_{in}} W[c_{out}, c_{in}, 0, 0] \sum_{i=0}^{3} |h_{in} - h_{in}^i||w_{in} - w_{in}^i| F_{in}[n, c_{in}, h_{in}^i, w_{in}^i] \\
&= \sum_{i=0}^{3} \sum_{c_{in}} W[c_{out}, c_{in}, 0, 0] |h_{in} - h_{in}^i||w_{in} - w_{in}^i| F_{in}[n, c_{in}, h_{in}^i, w_{in}^i] \\
&= \sum_{i=0}^{3} |h_{in} - h_{in}^i||w_{in} - w_{in}^i| \sum_{c_{in}} W[c_{out}, c_{in}, 0, 0] F_{in}[n, c_{in}, h_{in}^i, w_{in}^i] \\
&= \sum_{i=0}^{3} |h_{in} - h_{in}^i||w_{in} - w_{in}^i| conv(F_{in}, W)[n, c_{out}, h_{in}^i, w_{in}^i] \\
&= f_\uparrow(conv(F_{in}, W))[n, c_{out}, h_{out}, w_{out}],
\end{aligned}
\tag{7}
$$

where $h_{in} = h_{out}/H_{out} \times H_{in}$ and $w_{in} = w_{out}/W_{out} \times W_{in}$. h_{in}^i and w_{in}^i is calculated as follows:

$$
\begin{aligned}
h_{in}^0 &= \lfloor h_{in} \rfloor, w_{in}^0 = \lfloor w_{in} \rfloor; h_{in}^1 = \lceil h_{in} \rceil, w_{in}^1 = \lfloor w_{in} \rfloor \\
h_{in}^2 &= \lfloor h_{in} \rfloor, w_{in}^2 = \lceil w_{in} \rceil; h_{in}^3 = \lceil h_{in} \rceil, w_{in}^3 = \lceil w_{in} \rceil.
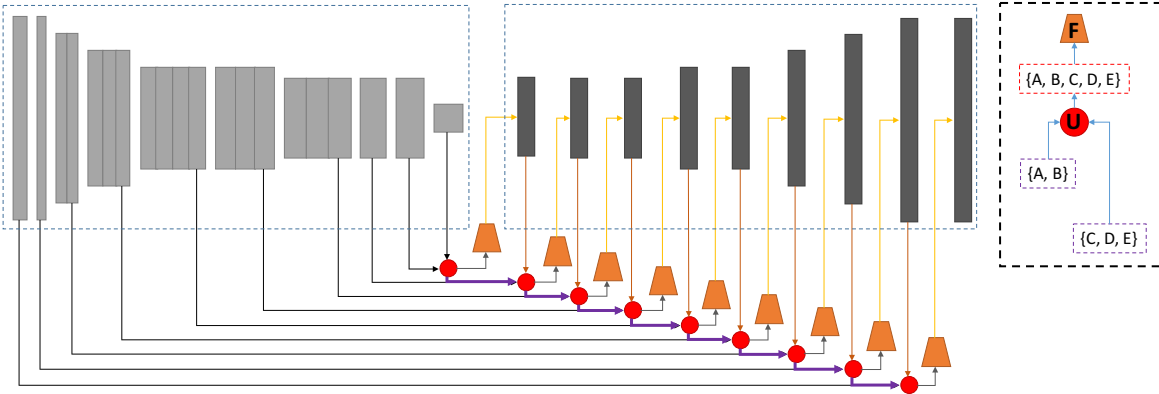\end{aligned}
\tag{8}
$$

$\square$

Figure 7: **Fully Dense Network with MobileNet V2 [32].** The inputs to the red circle (U) are multiple sets of features, while the output is the union of all the sets. **F** is the decoder stage, which takes a set of features as input. Best viewed in color.

Fully Dense Network with MobileNet V2

Figure 7 presents the Fully Dense Network based on MobileNet V2. The inputs to the red circle (U) are multiple sets of features, while the output is the union of all the sets. **F** is the decoder stage, which takes a set of features as input.