Expert Systems With Applications 193 (2022) 116438

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Pixel Voting Decoder: A novel decoder that regresses pixel relationships for segmentation

Pengfei Xian [a,*], Lai-Man Po [a], Jingjing Xiong [a], Chang Zhou [a], Yuzhi Zhao [a], Wing-Yin Yu [a], Weifeng Ou [a], Yujia Zhang [a], Xiaori Zhang [b]

[a] City University of Hong Kong, Hong Kong Special Administrative Region
[b] Fudan University, China

ARTICLE INFO

Keywords:
Convolutional neural network
Dynamic deconvolution
Encoder-Decoder
Image segmentation
Pixel voting
Residual block

ABSTRACT

With the rapid development of the convolutional neural network, both instance segmentation and semantic segmentation have achieved remarkable performances. Recently, many efforts have been made to use a unified Encoder-Decoder architecture to solve these two segmentation tasks simultaneously. The encoder extracts high-level features from the input images for both tasks. However, existing decoders cannot meet the performance requirements of these two tasks: the semantic segmentation decoder is not flexible enough for instance segmentation, and the instance segmentation decoder lacks the precision of semantic segmentation. Therefore, we introduce a novel Pixel Voting Decoder to satisfy both precision and flexibility. The proposed decoder regresses the interlayer pixel relationships between the input and output feature maps across the convolutional layers. Then, the pixel relationships are regarded as the pixel votes for dynamically decoding the higher level information from the encoder. Finally, we propose the dynamic deconvolution to make full use of the votes for each pixel during the decoding process. Meanwhile, the matrix computation for the dynamic deconvolution is designed to boost the calculation. Experiments show that the proposed method can achieve better performance than the well-known methods on both instance segmentation on the COCO dataset and semantic segmentation on the Cityscapes dataset. The matrix implementation of the dynamic deconvolution also shows its high efficiency and feasibility.

1. Introduction

IMAGE segmentation task aims to partition the digital images into multiple meaningful subregions and classify them. It mainly includes two sub-tasks: instance segmentation (Hafiz and Bhat, 2020) and semantic segmentation (Lateef and Ruichek, 2019). Instance segmentation firstly detects each object in the image, then marks the pixels occupied by each object with a label as the segmentation result. Semantic segmentation classifies all the pixels in the image, and the classification marks of each pixel constitute the segmentation result. Instance segmentation can distinguish different objects in the same category, but it does not mark pixels in regions where no objects are detected. For example, as shown in the left part of Fig. 1, dogs in the image will be marked with different labels, while the pixels with no object covered will not be labeled. On the contrary, semantic segmentation provides a label for each pixel but cannot distinguish between different instances. In other words, the pixels of two nearby dogs are all marked with the same "dog" label in semantic segmentation, and it cannot separate the dogs from each other by the semantic segmentation masks, as shown in the right part of Fig. 1.

Due to the rapid development of the convolutional neural networks, when using the Encoder-Decoder (Badrinarayanan et al., 2017) neural network with different network designs, both segmentation tasks now have much better performance than traditional image processing methods. In recent years, some efforts (Kirillov et al., 2019; Xiong et al., 2019; Cheng et al., 2020:) have tried to use a general Encoder-Decoder architecture to perform both semantic segmentation and instance segmentation at the same time. However, it is difficult to perform these two segmentation tasks using the same framework. So far, the existing encoders can be easily shared for both tasks, e.g., a pre-trained ResNet-50 (He, 2016), while the existing decoders are difficult to achieve good performances on these two tasks simultaneously.

On the one hand, the design of existing decoders can achieve excellent performance in the areas they target. The existing semantic

* Corresponding author at: City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong Special Administrative Region.
E-mail address: xian.pf@my.cityu.edu.hk (P. Xian).

https://doi.org/10.1016/j.eswa.2021.116438
Received 1 May 2021; Received in revised form 13 September 2021; Accepted 19 December 2021
Available online 31 December 2021

Fig. 1. Comparison of instance segmentation (left) and semantic segmentation (right). Instance segmentation can distinguish different dogs in the same category and labels are only assigned to the pixels within the detected objects. Semantic segmentation can assign a label for every pixel.

segmentation decoders can generate high precision segmentation results. These decoders are usually designed based on the feature fusion pyramid (Long et al., 2015). Full scales of feature maps are enlarged and added together down along the feature pyramid. Each feature level in the encoder is delivered to the corresponding decoding entrance of the pyramid. This design implicitly requires 3 factors that are hard for instance segmentation to satisfy. 1. The decoder should receive all levels of the feature maps from the encoder; 2. The pixels across the decoding convolutional layers should be aligned with similar patterns in the encoder; 3. Each feature map from the encoder should be fed into the specific decoder entrance to match the corresponding parameters. The existing instance segmentation decoders can achieve the flexibility to process different kinds of masks. These decoders are always designed as the downstream of object detections, they employ a simple fully convolutional network to decode single-level features. Similarly, this design also compromises 2 implicit limitations that prevent the existing semantic segmentation decoders from organizing multi-level features with the pyramid. 1. The objects are detected from different feature levels, so, the relative feature maps are not suitable for a pyramid for some detected objects. 2. The objects are detected at different positions on a feature map, making it difficult to align features at adjacent levels. The instance segmentation decoders choose to decode based on single-level features, making it flexible enough to process all kinds of detected objects.

On the other hand, the existing decoders cannot meet the flexibility and precision requirements at the same time. The existing semantic segmentation decoders perform poorly on organizing all kinds of features for the detected objects. That is because the feature fusion pyramid is only suitable for few objects that are detected at specific positions. Directly feeding the features into the pyramid may cause non-convergence problems due to the mismatch of features and parameters. The existing instance segmentation decoders can hardly achieve high precision because the decoding is only based on a single feature level. Besides, the fully convolutional network emits very small sizes of features. For example, Mask R-CNN (He, 2017), the most famous instance segmentation solution, produces segmentation masks with only 14×14 sizes, then directly interpolates them up to the same sizes of the original images, 1333×800, which will lose lots of precision.

In order to address the above problems, we propose a novel decoder, namely **Pixel Voting Decoder**. It simulates the pixel relationships between two feature maps across a convolutional block in the encoding process, then makes use of these pixel relationships to decode the feature maps and obtain the segmentation masks. The pixel relationships record how pixels in a feature level are related to the pixels across the convolutional layers. By utilizing the pixel relationships, pixels at different levels can be tracked and aligned, all levels of the feature maps can be employed and organized, the flexibility and precision can be achieved at the same time. Usually, each pixel is related to multiple pixels, we call the process that multiple relative pixels are summed weighted by the pixel relationships as pixel voting.

Similar to most of the encoder-decoder architecture, the Pixel Voting Decoder also iteratively uses the repeated basic block, as demonstrated in Fig. 2. The repeated block can be divided into 2 parts: the pixel association module and the dynamic deconvolution module. The pixel

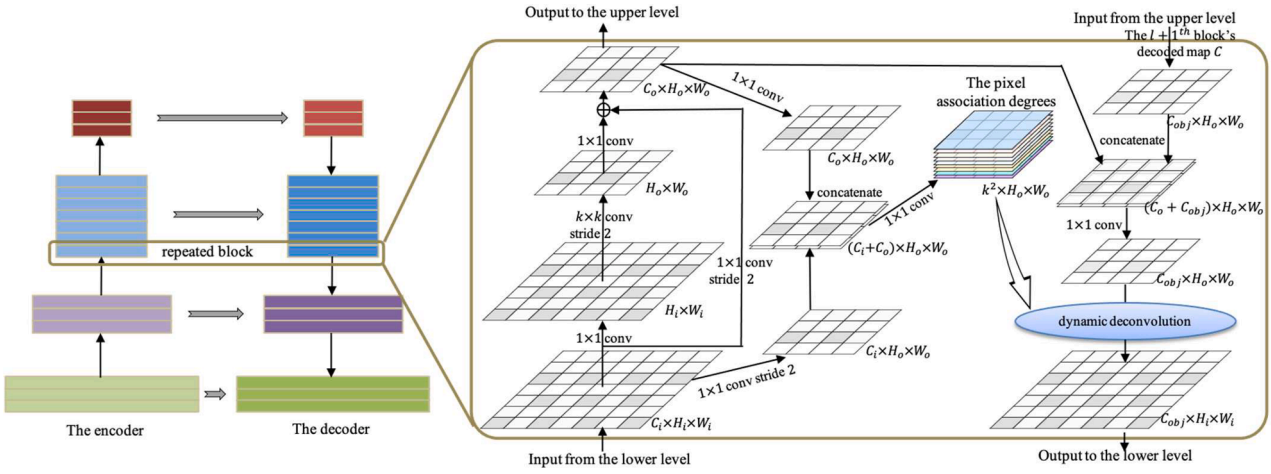
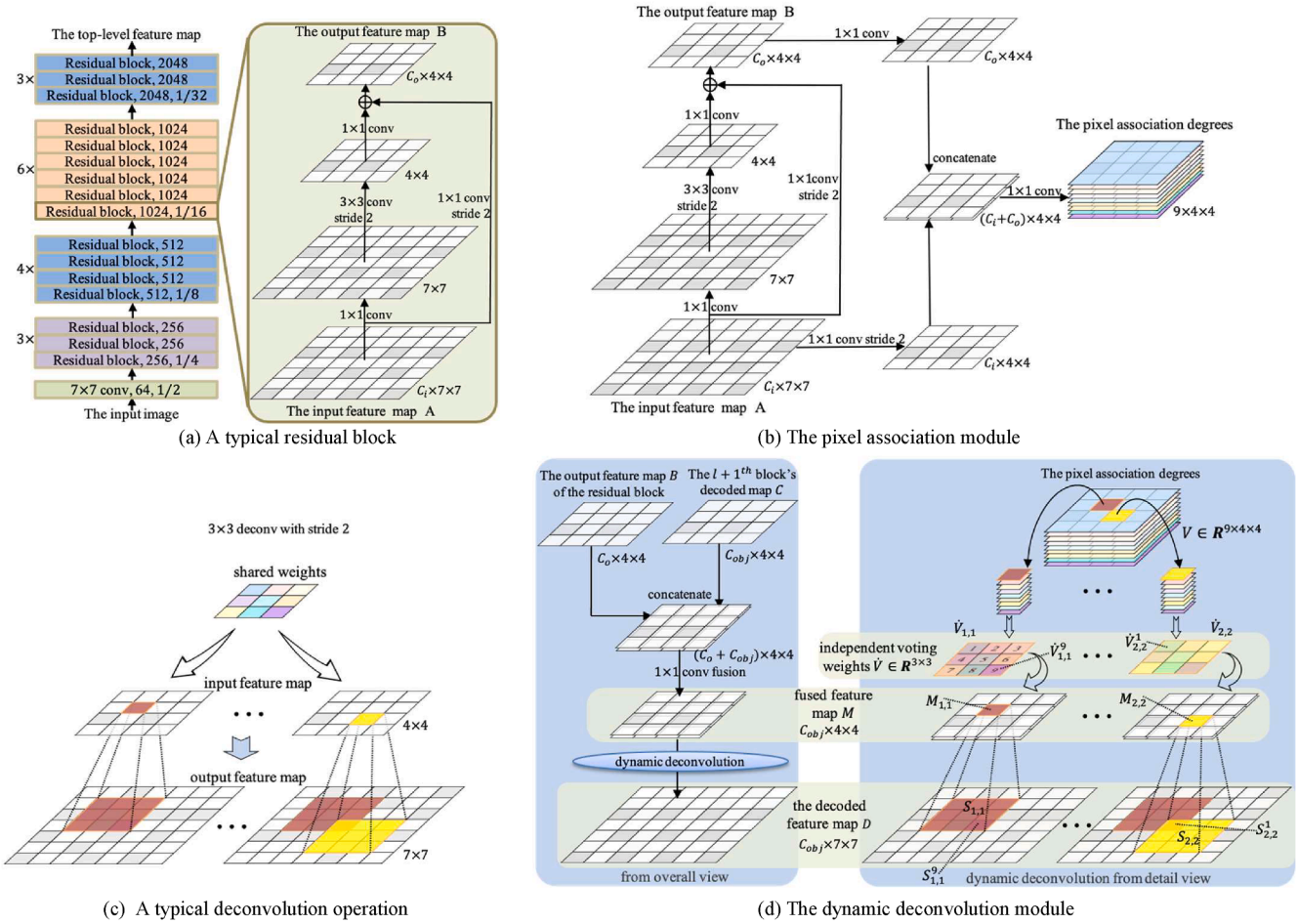


Fig. 2. The overview of the Pixel Voting Decoder, which is based on the encoder-decoder architecture. Both the encoder and the decoder consist of iterative basic blocks. The major contributions are in the basic blocks. Take a basic block as an example, the input feature passes through a residual block and obtains the output feature. Then, the input and output features are concatenated and convolved with 1×1 convolutional layer to regress the pixel association degrees. For decoding, the output feature and the upper level's decoded input are concatenated and fed into the dynamic deconvolution module to obtain the decoded feature map. The marks $\boldsymbol{H_i}$ and $\boldsymbol{W_i}$ refer to the input feature map height and width sizes, while $\boldsymbol{H_o}$ and $\boldsymbol{W_o}$ refer to the output map size. $\boldsymbol{C_i}$ and $\boldsymbol{C_o}$ denotes the channel number of input and output. $\boldsymbol{C_{obj}}$ denote the number of detected objects for the instance segmentation or the number of categories for semantic segmentation.



(a) A typical residual block

(b) The pixel association module

(c) A typical deconvolution operation

(d) The dynamic deconvolution module

Fig. 3. The overview of the Pixel Voting Decoder. (a) illustrates a typical residual block of the ResNet-50, where C_i and $C_o$refer to the channel number of the input and output maps, 7×7, 4×4 indicates the example feature map sizes, the stride equals 2. Residual blocks are repeatedly employed to reduce the feature map size and produce high-dimension features level by level. A residual block receives an input feature map A and convolves for 3 layers with 1×1, 3×3, 1×1 kernels in sequence. The convolution result is elementally added to map A with a bypass 1×1 convolution to fit the feature size and channel number. The added result serves as the residual block's output feature map B. (b) demonstrates the pixel association module based on a residual block. The input and output feature map A and B of the residual block are concatenated after a 1×1 convolution respectively, the concatenated feature map produces the pixel association degrees (a 9-channel tensor) using a convolution layer for regression. (c) illustrates a typical 3×3 kernel deconvolution operation with stride 2. Take 2 pixels in the input feature map as examples, in deconvolution, the pixels at different locations elementally multiply with the shared 3×3 weights, the 3×3 result pixels are added to the corresponding locations in the output feature map. (d) demonstrates the dynamic deconvolution module, the decoder part of the network, from overview and detail view, respectively. The left part is from the overview. The output feature map B from the corresponding residual block is concatenated with the output decoded feature map C of the upper-level residual block's decoder part, forming a fused feature map M, then processed by the dynamic deconvolution, the output decoded feature map D is obtained. The right part is from a detail view. The dynamic deconvolution is similar to the typical deconvolution, except that it abandons shared weights, instead, it employs the 9 values at the corresponding pixel in the pixel association degrees to serve as the deconvolution weights.

association module is designed to simulate the pixel relationships and regress *the pixel association degrees* across the convolutional layers. The dynamic deconvolution module utilizes the pixel association degrees to deliver the higher-level features to the lower level and fuse the features to generate segmentation masks. Because the traditional deconvolution is not suitable for the pixel voting scheme, we propose the dynamic deconvolution and design the matrix computation to boost the calculation.

We evaluate the proposed Pixel Voting Decoder for semantic segmentation and instance segmentation tasks on the Cityscapes dataset and the COCO dataset, respectively. Compared with the previous pipelines (Chen, et al., 1706; He, 2017), the proposed decoder has better performance than the popular methods on both tasks. In addition, we also visualized some samples on the Cityscapes and COCO datasets. In summary, the main contributions of this paper are as follows:

(1) This paper reveals the observation that the existing decoders cannot satisfy the instance segmentation and semantic segmentation tasks at the same time.
(2) We propose a network Pixel Voting Decoder. It can achieve good performance for these two segmentation tasks. It regresses the pixel relationships across the convolutional layers, then uses the pixel relationships to obtain better segmentations.
(3) We propose the dynamic deconvolution. It makes full use of the pixel relationships to fuse and decode the features. It dynamically deconvolves the features and uses the pixel relationships as the kernel weights.
(4) We implemented the matrix computation of the dynamic deconvolution to increase the calculation speed and reduce the memory cost.
(5) We evaluated the proposed Pixel Voting Decoder for semantic segmentation and instance segmentation tasks. Then compare it with well-known methods to demonstrate that better performance can be obtained.

2. Related works

2.1. The encoder – Decoder architecture

The Encoder-Decoder framework (Badrinarayanan et al., 2017) is widely used for image segmentation tasks. The framework contains two functional modules: the encoder and the decoder. The encoder, represented by an encoding function $z = f(x)$, down-samples the input image into high-dimensional latent-space feature maps through convolutional layers level by level, thereby reducing the feature map, increasing the channel number, and generating the high-dimensional feature maps are generated. The encoder usually emits 4 layers of feature maps, customarily marked from level 2 to level 5. The higher the level, the smaller the feature map and the more macroscopic information the feature map contains. Widely used encoders include: AlexNet (Krizhevsky et al., 2012), VGGNet (Simonyan and Zisserman, 2014), ResNet (He, 2016), GoogLeNet (Szegedy et al., 2015), MobileNet (Howard et al., 1704), and DenseNet (Huang et al., 2017), etc.

The purpose of the decoder, $y = g(z)$, aims to up-sample the decoded output feature map from the adjacent higher-level residual block utilizing deconvolution (Noh et al., 2015) or unpooling (Škrabánek, 2016), then aggregate the expanded feather map with the corresponding level's feather map from the encoder. Long et al. (Long et al., 2015) firstly proposed to up-sample the higher-level feature maps and aggregate them with the lower features using a fully convolutional network (FCN). FCN up-samples the feature maps by means of direct bilinear interpolation. On the contrary, Badrinarayanan et al. (Badrinarayanan et al., 2017) proposed the SegNet to replace the interpolation with unpooling. The SegNet uses pooling indices from the max-pooling step in the encoder to perform non-linear up-sampling. Ronneberger et al. (2015) proposed the U-Net, which provides a symmetric up-sampling decoder to the encoder so that the decoder can make full use of all levels of feature maps. Zhao et al. (Zhao, 2017) developed the Pyramid Scene Parsing Network (PSPNet), a multi-scale network to better learn the global context representation of a scene. Chen et al. (Noh et al., 2015; Chen et al., 1412; Chen et al., 2017) proposed the DeepLab series by employing the dilated convolution on the hidden features to improve the performance. The decoders stated above are mostly employed in semantic segmentation tasks. As for instance segmentation, He et al. (He, 2017) proposed Mask R-CNN, whose decoder can align the arbitrary size of cropped features into a unified size 14×14, so as to satisfy the flexibility requirement of instance segmentation.

2.2. Instance segmentation and semantic segmentation

The networks of instance segmentation and semantic segmentation, especially the decoder parts, are designed based on very different ideas. Semantic segmentation methods mainly focus on increasing the perception of hidden features: Zhao, et al. (Zhao, 2017) perform spatial pyramid pooling at several grid scales and improves the precision. Chen et al. (Chen et al., 1412) and Dai et al. (Dai, 2017) enlarge the perceptual fields for the convolutional kernels by introducing the dilated convolution and deformable kernels. Instance segmentation mainly extends the Mask R-CNN (He, 2017) to improve the performance. Bolya et al. (Bolya et al., 2019:) proposed the YOLACT to further crop and assemble the object masks to improve the Intersection over Union (IoU) performance. Huang et al. (Huang et al., 2019:) proposed to add an extra mask IoU for calculating the loss which improves the IoU performance. Recently, Kirillov et al. proposed another subtask, the panoptic segmentation, to solve both instance segmentation and semantic segmentation tasks using the same network. Many works (Xiong et al., 2019:; Cheng et al., 2020:) tried to solve this novel task using the encoder-decoder architecture. They employ the same encoder for extracting the features, however, they promise to use different decoders to deal with the two tasks.

2.3. Deconvolutions

The deconvolution, a.k.a. the transposed convolution, aims to swap the forward and backward passes of a convolution (Zeiler, 2010), which is always used to restore the feature size of a feature map that has passed through a convolution. Zeiler et al. (Zeiler et al., 2011) first proposed the deconvolution networks. Noh et al. (2015) proposed that deconvolution is a mirrored version of the convolution, which associates a single input activation with multiple outputs. Vincent et al. (Zeiler, 2010) employ a set of the shared kernel weights to sweep over the input feature map, each pixel is multiplied with the elements in the kernel, the kernel products are then summed up according to their pixel positions. As far as we know, the existing deconvolutions all use the shared kernel during the sweeping process.

3. Pixel Voting Decoder

3.1. Problem formulation

The encoder-decoder architecture is widely used in segmentation tasks, and our network also employs this architecture. As shown in Fig. 3 (a), the encoder uses the residual block to perform a level-by-level down-sampling operation, it reduces the feature map size and generates high-dimensional feature maps. The higher the level, the more comprehensive the extracted image features, and the smaller the size of the feature map. Symmetrical to the encoder, the decoder, shown in Fig. 3 (d), performs a level-by-level up-sampling operation. The encoder-decoder runs based on the repeated basic blocks. The residual blocks are the repeated basic blocks of the encoder, and similarly, the dynamic deconvolution module is the repeated basic block of the decoder. The residual block receives the input feature map A and sends out an output feature map B. Then, the dynamic deconvolution module receives the map B and the feature map C, and generates the decoded feature map D. Map C is the decoded feature map from the decoder of the upper-level residual block. Map B and map C have the same feature map size, meanwhile, map A and map D have the same feature map size. Suppose map C contains the higher-level coarse segmentation information, the decoder can accurately propagate it to the finer-segmented map D of larger sizes.

In detail, as shown in Fig. 3, the residual block is sequentially convolved in 3 layers with $1 \times 1, k \times k, 1 \times 1$ kernels on the input feature map A. The convolution result is elementally added to feature map A with a bypass 1×1 convolution to fit the feature size and the number of channels. The added result is served as the output feature map B. Note that k refers to the kernel size of the middle convolutional layer, generally $k = 3$. Then, the feature map A and B are concatenated after a 1×1 convolution, and a convolutional layer is used to regress the concatenated feature map into the pixel association degrees. The dynamic deconvolution module receives the decoded feature map C from the upper module, then aggregates it with the output feature map B from the corresponding level residual block in the encoder. Feature maps C and B are concatenated as the input of the dynamic deconvolution. The dynamic deconvolution is parameterized by the pixel association degree, and the decoded feature map D of the current module is sent out. Repeat this process level by level, and finally, the bottom up-sampling feature map will act as the final output segmentation masks. The process described above can be expressed by

$$Z = Enc(X) \tag{1}$$

$$Y = Dec(Z) \tag{2}$$

where X, Y, Z represent the original input image, the final output segmentation masks, and the embedded feature maps, respectively. Moreover, *Enc* and *Dec* represent the encoder and the decoder, respectively. In our case, the *Enc* is exactly the backbone of ResNet-50 (He, 2016).

The encoder and decoder are designed in an iterative approach, and the details of (1) and (2) can be expressed by iterative formulas. From a detail point of view, the encoder consists of several repeated residual blocks. If the total level number is L, we have

$$z_{l+1} = res_l(z_l)$$
$$Z = \{z_1, \cdots, z_L\} \tag{3}$$

where $l \in 0, \cdots, L-1$ denotes the level index in the encoder, res_l represents the l^{th} level residual block, z_l and z_{l+1} represent the extracted hidden feature maps. For each residual block, z_{l+1} denotes the output feature map B, while z_l refers to the input feature map A, which is also the output feature map B of the $l-1^{th}$ level residual block. We define z_0 as the original input image X. The embedded feature maps Z contains all of the encoder's output feature maps from z_1 to z_L.

The dynamic deconvolution module receives two inputs and produces one output. The iterative decoding process can be expressed as

$$y_l = dyd_l(y_{l+1}, z_{l+1}) \tag{4}$$

where dyd_l denotes the l^{th} level dynamic deconvolution module, y_l represents the output decoded feature map of the current level, which is expressed as the map D in Fig. 3, while y_{l+1} represents the upper-level output decoded feature map. At the top level, there is no upper level, so we define $y_L = z_L$. The output segmentation mask Y is exactly y_0. By introducing the above iterative representations, we have formalized an encoder-decoder network. The entire network can be divided into several repeated blocks, including the two modules. The pixel association module corresponds to equation (3) and the dynamic deconvolution module corresponds to equation (4). Both of these modules act as iterators, so they will be reused at all levels.

3.2. The pixel association regression

The primary task of Pixel Voting Decoder is to model the pixel voting relationship between layers, based on this setting, it is important to regress the pixel association degrees firstly for each level. The pixel association module is designed based on the residual block because the residual block is widely used in the cutting-edge encoders (Peng, 2019; Szegedy et al., 2015; Howard et al., 1704; Huang et al., 2017). Also, it meets the iterative repeating design described in formula (3) very well. Fig. 3 (a) illustrates the residual block of ResNet-50, where C_i and $C_o$refer to the channel number of the input and output maps, 7×7 and 4×4 indicates the example feature map sizes, the stride is 2. The residual blocks are repeatedly employed to reduce the feature map size and produce high-dimensional features level by level. For each residual block, there is only one convolutional layer whose kernel size k is larger than 1. So, depending on the middle convolutional layer kernel size k, each pixel in map A is only associated with no more than $n = k^2$ pixels in map B.

Based on the above observations, the Pixel Voting Decoder regresses the pixel association degrees between map A and map B. As demonstrated in Fig. 3 (b), the regression is conducted in the pixel association module. After convolving both maps with a 1×1 kernel respectively, the module concatenates the input map A and output map B across the residual block, then employs a convolutional layer to regress the association degrees from each pixel in map B to n pixels in map A. The regressed pixel association degrees record the pixel voting relationships, which play a key role in the successive dynamic deconvolution module.

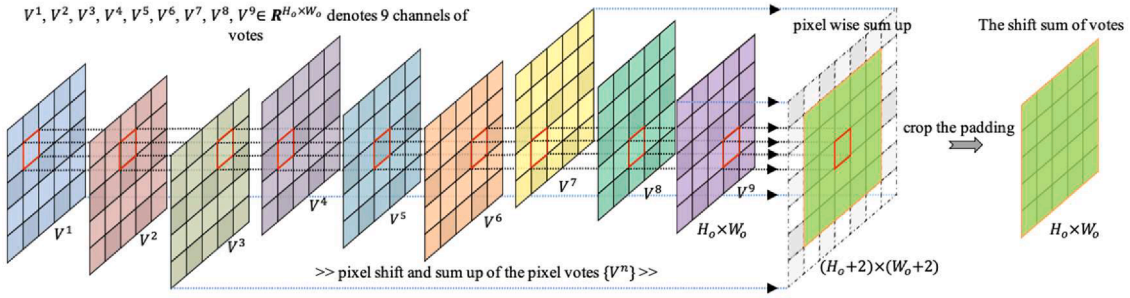
Here is a demonstration to make the network flow easier to understand, by means of following the above process from the perspective of a tensor. Suppose a tensor with size $C_i \times 7 \times 7$, where C_i denotes the channel number, is the input feature map A of a residual block. The stride of the residual block is 2, the middle convolutional layer kernel size k is 3, so, after 3 stacks of convolutional layers, a bypass convolution, and the elementary sum in the residual block, we get an output feature map B, a tensor with size $C_o \times 4 \times 4$. Map B is the basic output of the residual block in the ResNet backbone. Then, the tensors map A and map B are processed with a kernel 1×1 convolutional layer respectively. Note that the convolution for map A is configured as the same stride as the residual block, which means after the convolution, the new map A has a size of $C_i \times 4 \times 4$, the new map B is still in size of $C_o \times 4 \times 4$. Both the new map A and B are concatenated into a tensor with size $(C_i + C_o) \times 4 \times 4$. The concatenated tensor is then convolved to obtain the result tensor with a size of $n \times 4 \times 4$, which is the pixel association degrees. For the case of $k = 3, n = 9$. So, there are n values at each pixel, they represent the dynamic kernel weights at each pixel position in the deconvolution sweeping process.

3.3. The dynamic deconvolution

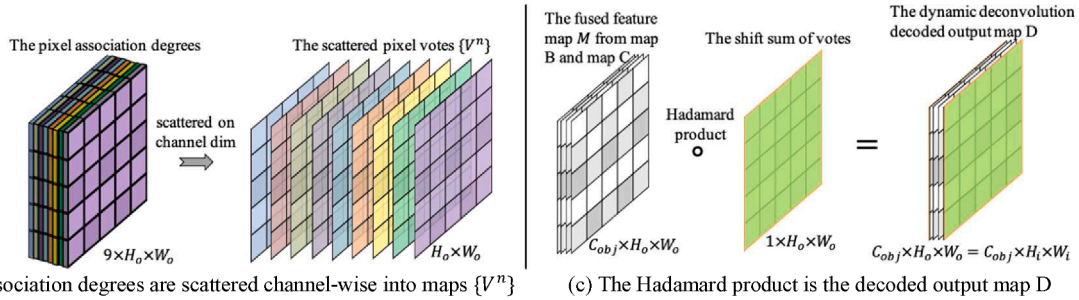
The dynamic deconvolution module is modified based on the traditional deconvolution (Mohammadi et al., 2016) operation to satisfy the requirement of our task. The typical deconvolution is the inverse operation of the convolution (Mohammadi et al., 2016). The convolution operation extracts the features from several pixels of the input map using a kernel filter and emits an output pixel. On the contrary, deconvolution associates one input pixel with several output pixels. Both the convolution and deconvolution operations are parameterized by a shared filter kernel. That is, during the kernel sweeping process, the kernel weights remain the same all over the feature map. Fig. 3 (c) illustrates a typical 3×3 kernel deconvolution operation with stride 2. Take 2 pixels in the input feature map as examples, in deconvolution, the pixels at different locations elementally multiply with the shared 3×3 weights, the 3×3 multiplication results are added to the corresponding pixel locations in the output feature map. The stride scheme in the deconvolution is similar to that in the convolution. In this way, each pixel in the input feature map is associated with n pixels in the output feature map.

However, in the deconvolution, the setting of sharing weight all over the feature map is unreasonable in our case, because the interlayer pixel relationships are different as the image content are changing during the deconvolution kernel sweeping process. To address this problem, the Pixel Voting Decoder proposes the dynamic deconvolution. That is, when the kernel sweeps over the feature map, at each pixel, there would be a set of unique weights in the kernel, the pixel association degrees at that position would act as the unique weights. We call this process the dynamic deconvolution. Fig. 3 (d) illustrates the decoder using the pixel association degrees and the dynamic deconvolution from the overall view and detail view. The left part is from the overview. The output feature map B from the corresponding residual block is concatenated with the output decoded feature map C from the $(l+1)^{th}$ level residual block's decoder. Then the concatenated feature maps are fused with a 1×1 convolutional layer, emitting the fused feature map M, afterward, processed by the dynamic deconvolution module, the output decoded feature map D is obtained. The right part is from the detail view. The dynamic deconvolution is similar to the typical deconvolution, except that it employs the pixel association degrees instead of shared weights as the deconvolution parameters. The pixel association degrees are organized as a n-channel tensor, which has the same map size as the map B, so, each pixel in the degree tensor contains n values, these n values in a pixel are reshaped into a $k \times k$ tensor and play the role of the dynamic deconvolution kernel weight. Then, the dynamic deconvolution can perform like a typical deconvolution operation to sweep all over the feature map with specific strides.

Follow the demonstration in the previous section, the network flow for the dynamic deconvolution module is described below. Two of the inputs for the dynamic deconvolution decoding, the output feature map B of the residual block and the regressed pixel association degrees, have been obtained. Another input is the feature map C with a size of $C_{obj} \times 4 \times 4$, which is actually the decoded output D from the $l+1^{th}$ level residual block's decoder, where C_{obj} denotes the tensor channel number,



(a) The per-channel feature maps $\{V^n\}$ are shifted 1 pixel, summed up pixelwise, and cropped the padding area.



(b) The pixel association degrees are scattered channel-wise into maps $\{V^n\}$ (c) The Hadamard product is the decoded output map D

Fig. 4. Demonstration of the process of matrix approach for Pixel Voting Decoder in case of stride = 1. The votes map $\boldsymbol{V}$ (a.k.a. the pixel association degrees) is scattered into k^2 per-channel vote maps $\{\boldsymbol{V}^n\}$ over the channel dimension. The per-channel maps $\{\boldsymbol{V}^n\}$ are shifted few pixels towards different directions and summed up pixelwise. The sum is then cropped the padding area and elementwise multiplied with the map $\boldsymbol{M}$ to obtain the decoded output feature map $\boldsymbol{D}$.

which equals to the number of detected objects in instance segmentation or the number of categories in semantic segmentation. The feature maps B and C are concatenated into a tensor with size $(C_o + C_{obj}) \times 4 \times 4$, then fused using an 1×1 convolutional layer into a $C_{obj} \times 4 \times 4$ tensor, which is the input of the dynamic deconvolution. The dynamic deconvolution employs the pixel association degrees ($n \times 4 \times 4$) as the deconvolution kernel weights, then emits the decoded feature map D with a size of $C_{obj} \times 7 \times 7$, which is the same map size as the input feature map A of the residual block. In detail, the pixel association degrees are transformed into $k \times k \times 4 \times 4$, at each pixel position during the deconvolution, there is a $k \times k$ filter that plays the role of deconvolution kernel. Note that map D also plays the role of input feature map C for the decoder of the $(l-1)^{th}$ level residual block. The decoding process is conducted level by level, residual block by residual block, at the bottom level ($l = 0$), the decoded map D is just the segmentation mask.

3.4. Acceleration: The matrix solution for the dynamic deconvolution

There is a problem in the deconvolution process. If the kernel weights are updated at every stop during the deconvolution kernel sweeping process, it will be very slow. The Pixel Voting Decoder turns to matrix operations to achieve fast and efficient implementation. Consider two typical strides: stride of 1 and stride of 2. We have designed the matrix computations for these two types of dynamic deconvolution.

Before demonstrating the matrix computation, we first analyze the pixel-level dynamic deconvolution module in detail. After concatenating the output feature map B of the residual block and the $(l+1)^{th}$ level decoded feature map C, the 1×1 convolution will mix the channel number into C_{obj}, and then obtain the fused feature map $M \in \boldsymbol{R}^{C_{obj} \times H_o \times W_o}$ as the input of the dynamic deconvolution. The regressed pixel association degrees act as the interlayer pixel voting map $V \in \boldsymbol{R}^{k^2 \times H_o \times W_o}$, which provides the values of the $k \times k$ kernel for each pixel position during the deconvolution sweeping process. As shown in the detail view of Fig. 3 (d), for each pixel $M_{i,j} \in \boldsymbol{R}^{C_{obj}}$ in the feature map M, there are k^2 votes $V_{i,j} \in \boldsymbol{R}^{k^2}$ at the corresponding pixel position in the feature map V. These votes are reshaped into a $k \times k$ kernel $\dot{V}_{i,j} \in \boldsymbol{R}^{k \times k}$ and multiplied with $M_{i,j}$ by broadcasting to obtain the single pixel voting result $S_{i,j} \in \boldsymbol{R}^{C_{obj} \times k \times k}$, which can be expressed as

$$S_{i,j} = M_{i,j} \circ \dot{V}_{i,j} \tag{5}$$

where $\circ$ denotes the Hadamard product. The center of $S_{i,j}$ is located at (i, j), covering k^2 pixels from $(i - \frac{k}{2}, j - \frac{k}{2})$ to $(i + \frac{k}{2}, j + \frac{k}{2})$, if indivisible, it will be rounded. The size mismatch between $M_{i,j}$ and $\dot{V}_{i,j}$ can be resolved by the broadcast mechanism. In detail, $M_{i,j}$ is reshaped into $C_{obj} \times 1 \times 1$, $\dot{V}_{i,j}$ is reshaped into $1 \times k \times k$, and the product $S_{i,j}$ has a shape of $C_{obj} \times k \times k$.

The single pixel voting results $\{S_{i,j}\}$ are added up with overlapping. In the dynamic deconvolution, the above process is conducted on each pixel by sweeping all over the feature map M, combining the deconvolution stride settings, the output feature map size is the same as the input feature map A of the residual block. Therefore, the decoded result of the l^{th} level is the overlapping sum $D \in \boldsymbol{R}^{C_{obj} \times H_i \times W_i}$, we have

$$D = \sum_{\substack{0 \le i < H_o \\ 0 \le j < W_o}} S_{i,j} = \sum_{\substack{0 \le i < H_o \\ 0 \le j < W_o}} M_{i,j} \circ \dot{V}_{i,j} \tag{6}$$

It is worth noting that the output feature size may change because of the stride, which is the same as the stride of the residual block, so that the feature size of feature map D is $H_i \times W_i$ instead of $H_o \times W_o$. For the case of stride = 1, the computation is simple. To better understand the overlapping sum, there is no harm in giving an example. Let $k = 3$, then, there are 9 pixels in the single pixel voting result $S_{i,j}$. As shown in Fig. 3 (d), we can number the pixels in $S_{i,j}$ with 1–9, the 3×3 voting results $\{S_{i,j}\}$ are overlapped with each other. For each pixel $D_{i,j}$ of the sum result, there are also 9 values from 9 voting results. Considering the overlapping bias, we have

$$\begin{aligned} D_{i,j} &= S^9_{i-1,j-1} + S^8_{i-1,j} + S^7_{i-1,j+1} \\ &+ S^6_{i,j-1} + S^5_{i,j} + S^4_{i,j+1} \\ &+ S^3_{i+1,j-1} + S^2_{i+1,j} + S^1_{i+1,j+1} \end{aligned} \tag{7}$$

where $S^n_{i,j}$ refers to the value at a certain relative pixel position in the 3×3 voting result, and its center is at $(i.j)$. For example, $S^9_{i,j}$ refers to the top

left value, while $S^9_{i,j}$ is the value of the bottom right value, and so on.

Note that $\{S_{i,j}\}$ cannot be represented as a feature map, because $\{S_{i,j}\}$ is a collection of 3×3 maps rather that single pixel values. However, $\{S^n_{i,j}\}$ can be represented as a feature map $S^n \in \boldsymbol{R}^{C_{obj} \times H_o \times W_o}$, and we have

$$S^n = \{S^n_{i,j}\} = M^\circ \{\dot{V}^n_{i,j}\} = M^\circ \dot{V}^n = M^\circ V^n \tag{8}$$

where $\dot{V}^n_{i,j}$ denotes the value at the corresponding relative pixel of the kernel $\dot{V}_{i,j}$, $\{\dot{V}^n_{i,j}\}$ denotes the collection of $\dot{V}^n_{i,j}$, $\dot{V}^n = \{\dot{V}^n_{i,j}\} \in \boldsymbol{R}^{H_o \times W_o}$ denotes the reconstructed map of the n-position pixels. V^n denotes the n^{th} channel feature map of V. For broadcasting, similarly, the $\dot{V}^n, V^n \in \boldsymbol{R}^{H_o \times W_o}$ are reshaped into $1 \times H_o \times W_o$.

Inspired by equations (7) and (8), it is easy to find that $D_{i,j}$ is the sum of the pixel-shifted version of the nearby voting result maps $\{S^n\}$. In addition, $D_{i,j}$ can represent the arbitrary pixel in the feature map D except the pixels at the edge of the feature map. As for the edge area, some values of the voting result map may be lost due to the pixel shift, which can be solved by padding zeros in the voting results. Denote $\overleftrightarrow{i}_{i,j}$ is an operation of shifting a feature map by i pixels towards the bottom by j pixels towards the right and padding zeros to the empty pixels. We can extend the equation (6) to the entire feature map as,

$$D = \sum_{n=1}^{9} \overleftrightarrow{S^n} = \overleftrightarrow{S^9}_{i-1,j-1} + \overleftrightarrow{S^8}_{i-1,j} + \overleftrightarrow{S^7}_{i-1,j+1} + \overleftrightarrow{S^6}_{i,j-1} + \overleftrightarrow{S^5}_{i,j} + \overleftrightarrow{S^4}_{i,j+1} + \overleftrightarrow{S^3}_{i+1,j-1} + \overleftrightarrow{S^2}_{i+1,j} + \overleftrightarrow{S^1}_{i+1,j+1} \tag{9}$$

Considering equation (8), we have

$$D = \sum_{n=1}^{9} M^\circ \overleftrightarrow{V^n} = M^\circ \sum_{n=1}^{9} \overleftrightarrow{V^n} = M^\circ (\overleftrightarrow{V^9}_{i-1,j-1} + \overleftrightarrow{V^8}_{i-1,j} + \overleftrightarrow{V^7}_{i-1,j+1} + \overleftrightarrow{V^6}_{i,j-1} + \overleftrightarrow{V^5}_{i,j} + \overleftrightarrow{V^4}_{i,j+1} + \overleftrightarrow{V^3}_{i+1,j-1} + \overleftrightarrow{V^2}_{i+1,j} + \overleftrightarrow{V^1}_{i+1,j+1}) \tag{10}$$

In this way, we designed a matrix method for the dynamic deconvolution of the Pixel Voting Decoder by only using matrix shifting and Hadamard product, we can obtain the output decoded feature map $D \in \boldsymbol{R}^{C_{obj} \times H_i \times W_i}$ in a much simpler way. Note that in case of stride $= 1$, $H_i = H_o$ and $W_i = W_o$. In order to show more intuitively, the above-mentioned matrix computation is illustrated in Fig. 4. In summary, the feature map V (a.k.a. the pixel association degrees) is scattered into k^2 per-channel vote maps $\{V^n\}$ over the channel dimension. The per-channel maps $\{V^n\}$ are shifted few pixels towards different directions and summed up pixelwise. Then the sum is cropped the extra padding area and elementwise multiplied with the feature map M to obtain the decoded output feature map D. The complex dynamic deconvolution is simplified into several matrix operations, which greatly increases the amount of computation. The consumption comparison is conducted in Section 4.4.

3.5. The matrix calculations for stride of 2

For the situation with stride of 2, the pixel relationships are somewhat complicated. Fig. 5 illustrates the pixel relationships between the input and output layers across a 3×3 convolutional layer with stride $=$ 2. For a convolutional layer with stride $= 1$, each pixel on the input map is always associated with 9 pixels in the output feature map when $k = 3$, while each pixel in the output feature map is associated with 9 pixels in the input map as well. But for stride $= 2$, even though pixels in the output map are still associated with 9 pixels from the input feature map, the pixels in the input feature map are no longer associated with as much as the 9 pixels in the output map. When dealing with dynamic deconvolution, it requires to precisely manage the interlayer pixel relationships.

However, if you follow the clue of interlayer pixel relationship between the layers, the pixel relationship becomes clearer. For the case where the stride is equal to 1, during the kernel sweeping process, each pixel in the input feature map can be covered by the center of the kernel. Now the stride is equal to 2, and only the pixels at the 2x2 stride position can be covered by the center of the convolution kernel. Look at the 4-pixels group, which consists of the kernel center covered pixel a, a's right neighbor pixel b, a's lower neighbor pixel c, and a's diagonally right lower neighbor pixel d. In such a 4-pixels group, the kernel center can be placed on the pixel a, but never on the pixels b, c, and d. The whole input map is a dense mosaic of the 4-pixels groups. We can focus on the 4-pixels group and study the pixel association patterns. For convenience, when the kernel center is on the input pixel a, the convolution output pixel is denoted as E, and the right, lower, and diagonally right lower neighbor of pixel E are denoted as the F, G and H, respectively.

As is shown in Fig. 5, pixel a is only associated with 1 pixel E instead of any other pixels in the output feature map. The pixel b is associated with 2 pixels E and F. The pixel c is associated with 2 pixels E and G. The pixel d is associated with 4 pixels A, B, C and D. The green, red, purple, and cyan shadows in the input map are the perception fields of the pixels E, F, G, H. The coverage of these perception fields indicates the pixel associations from a, b, c, d to E, F, G, H. There are only these 4 types of pixel association patterns for the case of stride equals 2. Any other pixel can be regarded as a translation version of one of the pixel association patterns.

Following the above observation results for the convolution process, we can infer the pixel association in the deconvolution process. Indicates that a, b, c, and d are now pixels in the output decoded feature map $D \in \boldsymbol{R}^{C_{obj} \times H_i \times W_i}$, E, F, G, H are the pixels in the fused feature map $M \in \boldsymbol{R}^{C_{obj} \times H_o \times W_o}$, according to the above observation, we can write down the pixel relationships as follow:

$$\begin{aligned} &a \rightarrow E \\ &b \rightarrow E,\ F \\ &c \rightarrow E,\ G \\ &d \rightarrow E,\ F,\ G,\ H \end{aligned} \tag{11}$$

Note that for the case where the stride is equal to 2, H_i and W_i are 2 times of H_o and W_o, so the number of pixels in the feature map D is 4

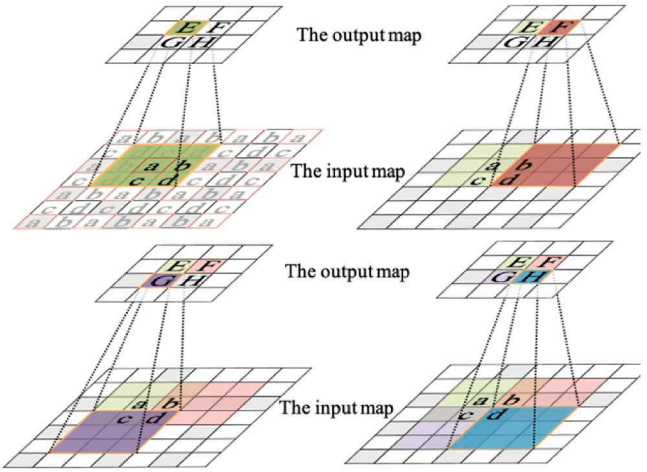


Fig. 5. The demonstration of the relationships between input and output layers across the convolutional layer with stride = 2. Pixels ***a***, ***b***, ***c***, ***d*** denotes 4 typical types of pixels in the input map, Pixels ***E***, ***F***, ***G***, ***H*** denotes adjacent pixels in the output map. The green, red, purple, cyan shadows in the input map are the perception fields of the pixels ***E***,***F***,***G***,***H***. From the coverage we find there are (***a*** vs. ***E***,), (***b*** vs. ***E***, ***F***), (***c*** vs. ***E***, ***G***), (***d*** vs. ***E***, ***F***, ***G***, ***H***) 4 types of interlayer pixel relationships.

times that of feature map M. Recall that the pixel voting map $V \in \boldsymbol{R}^{9\times H_o\times W_o}$ emitted from the pixel association module has 9 channels. Based on the pixel relationships, we can determine the output decoded feature map D as

$$
\begin{aligned}
D^a &= M^{\circ} \overleftarrow{V^1}_{i,j} \\
D^b &= M^{\circ}\left(\overleftarrow{V^2}_{i,j} + \overleftarrow{V^3}_{i,j-1}\right) \\
D^c &= M^{\circ}\left(\overleftarrow{V^4}_{i,j} + \overleftarrow{V^5}_{i-1,j}\right) \\
D^d &= M^{\circ}\left(\overleftarrow{V^6}_{i,j} + \overleftarrow{V^7}_{i,j-1} + \overleftarrow{V^8}_{i,j-1} + \overleftarrow{V^9}_{i-1,j-1}\right)
\end{aligned} \tag{12}
$$

where $\overleftarrow{V^1}_{i,j}$ still represents shifted votes. This is an implicit feature mapping, in which 9 voting channels are mapped in pixel relationships $[a{\to}E, b{\to}E, b{\to}F, c{\to}E, c{\to}G, d{\to}E, d{\to}F, d{\to}G, d{\to}H]$. D^{a-d} represents the corresponding pixels, as described in the input feature map in the upper left graph of Fig. 5. D^{a-d} has the same feature map size as the feature map M and the voting (the pixel association degrees) V, making it suitable for matrix operations. For PyTorch (Dumoulin and Visin, 2016), this operation is feasible and can be realized with the operations as

$$
\begin{aligned}
D[0::2,\ 0::2] &= M^{\circ} \overleftarrow{V^1} \\
D[0::2,\ 1::2\ ] &= M^{\circ}\left(\overleftarrow{V^2} + \overleftarrow{V^3}\right) \\
D[1::2,\ 0::2\ ] &= M^{\circ}\left(\overleftarrow{V^4} + \overleftarrow{V^5}\right) \\
D[1::2,\ 1::2\ ] &= M^{\circ}\sum_{n=6}^{9} \overleftarrow{V^n}
\end{aligned} \tag{13}
$$

where $[s::2]$ refers to the index of every 2 pixels picked up in M_{l-1} starting from s. As we discussed, most of the residual blocks only employ these two strides of convolutional layers. By analogy, the Pixel Voting Decoder modules for other special residual blocks can also be easily obtained. With the help of the matrix computations, dynamic deconvolution can achieve a speed comparable to the official PyTorch convolution implementation, which is much faster than the step-by-step value replacement operation.

Table 1
Semantic segmentation performance Comparison on Cityscapes dataset. Coarse: trained on fine annotation dataset plus extra coarse annotations dataset.

Method	Coarse	mIoU
DeepLabv2-CRF (Chen, et al., 2017)		70.4
ML-CRNN (Long et al., 2015)		71.2
Deep Layer Cascade (Ronneberger et al., 2015)		71.1
FRRN (Krizhevsky et al., 2012)		71.8
Adelaide_context (Szegedy et al., 2015)		71.6
FoveaNet (Howard et al., 1704)		74.1
LRR-4x (Huang et al., 2017)	✓	71.8
RefineNet (Chen et al., 1412)		73.6
Ladder DenseNet (Chen et al., 2017)		74.1
Global-local-Refinement (Noh et al., 2015)		77.3
PEARL (Škrabánek, 2016)		75.4
SAC_multiple (Dai, 2017)		78.1
SegModel (Shen, 2017)	✓	79.2
ResNet-38 (Bolya et al., 2019:)	✓	77.9
PSPNet (Zhao, 2017)		78.6
DeepLabv3 (Chen, et al., 1706)	✓	79.1
Ours		79.3

4. Experiments

4.1. Experimental settings

Datasets. In our experiments, the used well-known public semantic segmentation dataset of Cityscapes (Cordts, 2016) and instance segmentation dataset of COCO (Lin, 2014). The Cityscapes dataset provides 19 categories and is composed of 5,000 fine annotated images. These images are urban street scenes and focus on semantic understanding of common driving scenarios. Due to its diversity, it is one of the most challenging datasets for panoptic segmentation because it covers scenes from more than 50 European cities. It is divided into the training set (2975), validation set (500), and test set (1525). The Cityscapes dataset is used to verify the semantic segmentation task of our proposed Pixel Voting Decoder. The COCO dataset is large object detection and instance segmentation dataset. This dataset targets scene understanding and is mainly intercepted from complex daily scenes. The targets in the image are calibrated by precise segmentation annotations. It provides 80 categories, more than 330,000 images, of which 16,4062 are annotated, and the number of individuals in the whole dataset exceeds 1.5 million. It is divided into training set (82,783), validation set (40,504), and test set (40,775). In our experiments, the COCO dataset is used for instance segmentation comparison of Pixel Voting Decoder in our experiments.

Evaluation Metrics. In order to compare the performance with previous methods, we follow the literature and use the mean Intersection over Union (mIoU) and Average Precision (AP) to calculate the accuracy.

Implementation Details. The input resolution is set to 512 × 1024. The raw data is augmented by color jittering, horizontal flipping, random scaling, and cropping. The learning rate is set to 0.01 and decayed in a poly curve with a power of 0.9. The batch size is set to 4, and the number of training iterations is set to 90 k. All those hyper-parameters are determined by maximizing the performance on the Cityscapes validation set.

In the following section, we treat Deeplab (Chen, 2014; Chen, et al., 2017; Chen, et al., 1706) version 3 as the baseline method for semantic segmentation and Mask R-CNN as the baseline for instance segmentation because they provide impressive reproducible codes and baseline models. We will also report MACs and the total number of parameters. All results were tested on a single GTX 2080Ti platform.

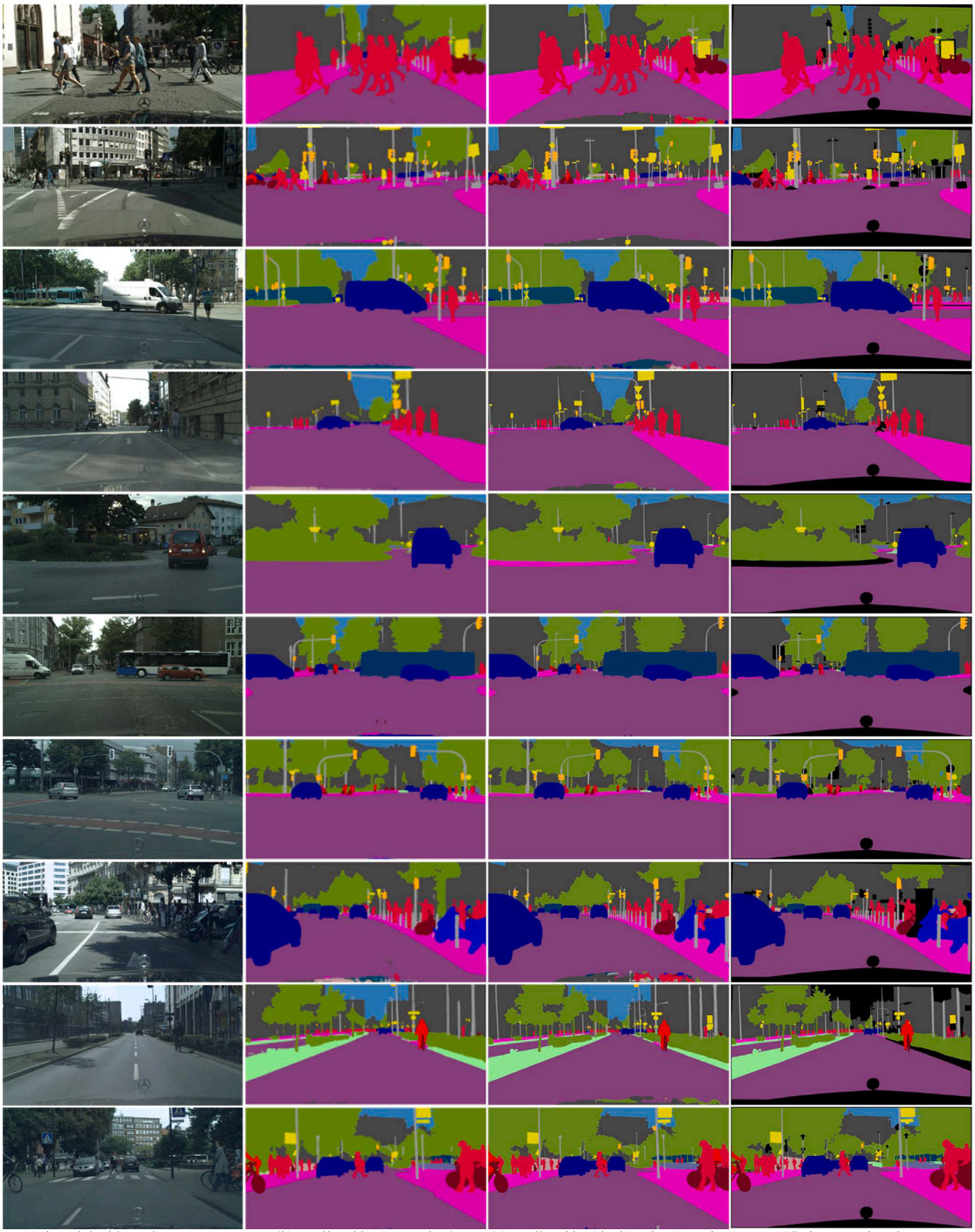


Fig. 6. The semantic segmentation visualization results on Cityscapes validation set compared with DeepLab v3. Column (a) is the original input images, column (b) is the semantic segmentation results predicted by DeepLab v3, column (c) is the semantic segmentation result predicted by our proposed Pixel Voting Decoder, column (d) is the semantic segmentation ground truth. Note that the black area in (d) refers to undefined labels.

Table 2
Instance segmentation performance comparison on COCO dataset with the mainstream Mask R-CNN.

Method	Backbone	AP	AP_{50}	AP_{75}
Mask R-CNN	Res-50-C4	30.3	51.2	31.5
Mask R-CNN	Res-50-FPN	33.6	55.2	35.3
Ours	Res-50-C4	33.9	56.1	36.2
PANet (Wang et al., 2020)	ResNeXt-101	42.0	62.3	46.4
SOLO (Xie et al., 2020:)	Res-DCN-101-FCN	40.4	62.4	43.7
MSRCNN (Huang et al., 2019:)	Res-DCN-101-FCN	38.3	58.8	41.5
Ours	ResNet-50-C4	33.9	56.1	36.2
PolarMask (Zagoruyko et al., 1604)	ResNet-101-FPN	32.1	53.7	33.1
YOLACT (Bolya et al., 2019:)	ResNet-101-FPN	29.8	48.5	31.2
MultiPath (Paszke, 2019)	ResNet-101	25.0	45.4	24.5

4.2. Validation on semantic segmentation

In order to verify the effectiveness of our proposed Pixel Voting Decoder, we adjust it into two tasks, namely semantic segmentation and instance segmentation. For semantic segmentation, we directly replace the feature pyramid decoder with our Pixel Voting Decoder. At the top of the ResNet backbone, for classification purpose, a 1×1 convolutional layer is employed to predict the class logits for the few pixels in the smallest-size feature map. From this classification layer, we obtain a tensor of shape $C_{obj} \times H_{top} \times W_{top}$ from the backbone of ResNet, where C_{obj} is equal to the number of semantic segmentation categories, and H_{top} and W_{top} represent the height and width of the feature map at the top encoder layer. Then, these top layer classification logits are regarded as the initial input feature map C, which is regarded as the y_L fed in Equation (4), finally, we obtain the decoded large mask with a shape of $C_{obj} \times H_{img} \times W_{img}$, where H_{img} and W_{img} refer to the height and width size of the original input image. Note that in our implementation, Pixel Voting Decoder only decodes the masks into $\frac{1}{4}H_{img} \times \frac{1}{4}W_{img}$ to reduce the huge computations for large feature maps. Eventually, $\frac{1}{4}H_{img} \times \frac{1}{4}W_{img}$ masks are unsampled into $H_{img} \times W_{img}$ by the bilinear interpolation.

We compared our Pixel Voting Decoder with the mainstream semantic segmentation methods (such as DeepLab, PSPNet, SegModel, etc.) on the Cityscapes validation dataset. We adapt the Pixel Voting Decoder to a typical encoder-decoder architecture. The performances of these networks are listed in Table 1. Note that the listed networks are trained using the ResNet-50 backbone with single scale inputs. As shown in the table, architecture with Pixel Voting Decoder achieves 79.3 on mIoU without the help of coarse annotations, which is better than most of the mainstream semantic segmentation methods. For a better comparison, we refer to the ground truth in Fig. 6 and visualize the semantic segmentation prediction results of our method and the most popular approach DeepLab v3. It is not difficult to find that the edges segmented by ours are sharper and the poles are connected properly. Note that the images of the DeepLab v3 are cropped from the original paper. The results show that our approach has a better detail segmentation performance.

4.3. Validation on instance segmentation

For the instance segmentation, we modified the mask head of Mask R-CNN (He, 2017) using the proposed Pixel Voting Decoder. The classification logits from the class head are employed as the primary mask for each object proposal. The pixel association degrees come from the backbone of ResNet, as shown in Fig. 3 (a). Considering that the number of the residual blocks is different due to the different source levels of the object proposals, we use the basic blocks of the decoder part of the Pixel Voting Decoders to process proposals from the encoder feature levels. By simulating dynamic deconvolution, we can obtain the cropped area of each object in all feature levels. Through similar operations in the semantic segmentation decoder, we finally obtain a large decoded mark with a shape $C_{obj} \times H_{img} \times W_{img}$, where C_{obj} refers to the number of object proposals from the Region Proposal Network (Konig, 2017) of Mask R-CNN.

In this section, we compare the Pixel Voting Decoder with the well-known instance segmentation method Mask R-CNN to prove its effectiveness. In order to control the effect of the Feature Pyramid Network (FPN), we employ ResNet-50-C4 as the backbone. We modify it by incorporating our Pixel Voting Decoder into Mask R-CNN, and the detailed architecture modification is described in Section 3.6. The

Table 3
The time consumption comparison of two implementations of the dynamic deconvolution, the step update version and matrix computation version on different strides and different input feature map sizes. The smaller is better.

Input map size (px)	Step update cost time (ms)		Matrix computation cost time (ms)	
	stride = 1	stride = 2	stride = 1	stride = 2
1024*1024	42,925	43,030	3.781	17.07
512*512	10,657	10,705	1.161	4.54
256*256	2,653	2,680	0.688	1.28
128*128	667.2	667.6	0.593	0.597
64*64	167.2	167.1	0.445	0.597
32*32	42.5	43.5	0.437	0.494
16*16	11.02	11.3	0.431	0.484
8*8	2.801	2.84	0.427	0.484

Fig. 7. The instance segmentation visualization results on COCO validation set. We mark the segmented objects with their contours. Texts in white indicate the predicted label for each object, the following figures indicate the confidence for each object.

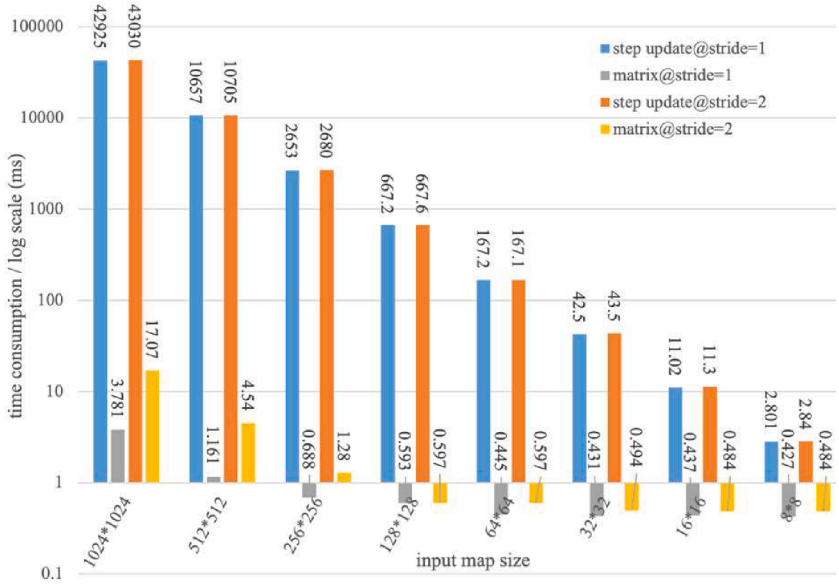


Fig. 8. The comparison of two implementations of the dynamic deconvolution at all sizes of the input feature maps in log scale. The matrix version is much faster than the step update version.

Table 4
The comparison between the dynamic deconvolution with the typical convolution and deconvolution upon the time consuming and the GPU memory cost when input map sizes = 1024*1024. The smaller is better.

Input map size	Time cost (ms)		GPU memory (MB)	
	stride = 1	stride = 2	stride = 1	stride = 2
PyTorch Conv2d	0.241	0.156	941	849
PyTorch ConvTransposed2d	0.292	0.534	941	1397
Dynamic deconvolution by step update	42,925	43,030	1011	1519
Dynamic deconvolution by matrix	3.781	17.07	987	1467

Instance segmentation performance is listed in Table 2. The results show that our Pixel Voting Decoder obtains higher metric scores than the vanilla ResNet-50-C4 backbone and FPN version, which shows that our Pixel Voting Decoder outperforms FPN. In order to intuitively demonstrate the performance of our Pixel Voting Decoder, some visualization results of instance segmentation are shown in Fig. 7. It is worth noting that the performance of the proposed Pixel Voting Decoder is conducted on a single GPU employing the ResNet-50 as the backbone. With limited computing resources, the proposed method achieves 33.9 on AP, which is better than the original Mask RCNN and the FPN variant, even better than some methods employing ResNet-101. Even though the more recent works (Wang et al., 2020; Xie et al., 2020:) set the pace on the metrics, the proposed method still achieves remarkable performance with fewer convolutional layers, which means less computation and less memory consumption.

4.4. Ablation Study: the efficiency of the matrix computation

As discussed in Section 3.4 and 3.5, the original dynamic deconvolution operation requires step-by-step updating of the deconvolution kernel weights during the kernel sweeping process. The proposed matrix computations can improve the calculation speed of the dynamic deconvolution. In this section, verify the efficiency of the matrix designed for dynamic deconvolution, and we demonstrate that the computational cost of the proposed dynamic deconvolution is affordable when compared with the typical convolution and deconvolution operations.

To compare the two implementations of the dynamic deconvolution, we conducted a series of experiments on various input feature map scales for different strides. The results show that no matter how the stride is set, the matrix implementation is much better than the original step update implementation by a large margin no matter how the stride is set. Besides, for both cases of stride 1 and stride 2, the matrix version demonstrates its supreme performances. Table 3 lists the detailed computation time of dynamic deconvolution using these two implementations. For an input feature map with a size of 1024*1024, the step update version costs over 40 s while the matrix computation version only uses 3.781 ms, the matrix version is over 11352x faster than the step update version. Fig. 8 provides a more intuitive perspective for comparing these two implementations on a log scale, the matrix version is 3 or 4 orders of magnitude faster on all scales of the input feature maps.

During the sweeping process, using different kernel weight settings at each stop will cost more computing resources, but compared with typical convolution and deconvolution operations, this is affordable. We compare the proposed dynamic deconvolution with the official PyTorch implementations of the convolution and deconvolution, the Conv2d and the ConvTransposed2d. The time consumption and the GPU memory cost results are listed in Table 4. We only analyze the matrix implementation of the dynamic deconvolution with the convolution and the deconvolution, because only this version is employed for the previous experiments. These methods are tested on a large 1024x1024 input feature map, the results show that the proposed dynamic deconvolution is a little slower than the typical operations, but the cost of stride 1 and stride 2 are only 4 ms and 17 ms, respectively. This is affordable for network training and inference. Note that the official convolution and deconvolution cannot update the kernel weights when sweeping the feature map. As for the GPU memory cost, the dynamic deconvolution consumes 987 MB for stride 1 and 1467 MB for stride 2, which is almost the same resources as the typical operations.

It is worth noting that the matrix computation of the dynamic deconvolution is implemented based on the tensor operation in python, there is still space for improving the computation speed, for example, using C++. Note that the input and output channel numbers of the above operations are set to 19. The cases of deconvolution when stride equals 2 cost more memories because their outputs are much larger, actually the size is 2048×2048.

5. Conclusion

In this paper, we proposed the Pixel Voting Decoder, which is a novel universal decoder for both semantic segmentation and instance segmentation tasks. This network regresses the interlayer pixel voting relationships across the convolutional layers, then utilizes the relationships to accurately deliver higher-level information down to the lower level and fuse the features to generate segmentation masks. We propose the dynamic deconvolution to adapt the deconvolution for pixel voting and design the effective matrix computation to boost the calculation. Compared with popular methods like Mask R-CNN and DeepLab, the proposed Pixel Voting Decoder showed its superiority in both flexibility and precision. The ability to decode the instance segmentation masks shows its flexibility, while the competitive performance for both tasks shows its precision. It can achieve better performance when dealing with the instance segmentation task on the COCO dataset and the semantic segmentation task on the Cityscapes datasets. The matrix version of the dynamic deconvolution also ensures the high efficiency of the proposed network. In the future, with more efficient implementation for the dynamic deconvolution, the network would run faster. Furthermore, the Pixel Voting Decoder will help to build a universal network targeting different segmentation tasks.

CRediT authorship contribution statement

Pengfei Xian: Conceptualization, Methodology, Software, Writing – original draft. **Lai-Man Po:** Supervision, Formal analysis, Writing – review & editing. **Jingjing Xiong:** Validation, Software, Writing – review & editing. **Chang Zhou:** Formal analysis, Writing – review & editing. **Yuzhi Zhao:** Resources, Writing – review & editing. **Wing-Yin Yu:** Writing – review & editing. **Weifeng Ou:** Writing – review & editing. **Yujia Zhang:** Data curation. **Xiaori Zhang:** Visualization, Writing – review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Badrinarayanan, V., Kendall, A., & Cipolla, R. (2017). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 39*(12), 2481–2495.

Bolya, D., Zhou, C., Xiao, F., et al. (2019). Yolact: Real-time instance segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 9157–9166).

L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, Semantic image segmentation with deep convolutional nets and fully connected crfs, arXiv preprint arXiv:1412.7062, 2014.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. L. (2017). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 40*(4), 834–848.

Chen, Liang-Chieh, et al., Semantic image segmentation with deep convolutional nets and fully connected crfs. arXiv preprint arXiv:1412.7062 (2014).

Chen, Liang-Chieh, et al. Rethinking atrous convolution for semantic image segmentation. arXiv preprint arXiv:1706.05587 (2017).

Chen, Liang-Chieh, et al. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. IEEE transactions on pattern analysis and machine intelligence 40.4 (2017): 834-848.

Cheng, B., Collins, M. D., Zhu, Y., et al. (2020). Panoptic-deeplab: A simple, strong, and fast baseline for bottom-up panoptic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 12475–12485).

Cordts, M., et al. (2016). The cityscapes dataset for semantic urban scene understanding. *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Dai, J., et al. (2017). Deformable convolutional networks. In *2017 IEEE International Conference on Computer Vision (ICCV)* (pp. 764–773).

Dumoulin, Vincent, and Francesco Visin. A guide to convolution arithmetic for deep learning. arXiv preprint arXiv:1603.07285 (2016).

Hafiz, A. M., & Bhat, G. M. (2020). A survey on instance segmentation: State of the art. *International Journal of Multimedia Information Retrieval*, 1–19.

He, K., et al. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*.

He, K., et al. (2017). Mask r-cnn. *Proceedings of the IEEE international conference on computer vision*.

A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications, arXiv preprint arXiv:1704.04861, 2017.

Huang, Z., Huang, L., Gong, Y., et al. (2019). Mask scoring r-cnn. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 6409–6418).

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4700–4708).

Kirillov, A., He, K., Girshick, R., et al. (2019). Panoptic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 9404–9413).

Konig, D., et al. (2017). Fully convolutional region proposal networks for multispectral person detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097–1105).

Lateef, F. H., & Ruichek, Y. (2019). Survey on semantic segmentation using deep learning techniques. *Neurocomputing, 338*, 321–348.

Lin, T.-Y., et al. (2014). Microsoft coco: Common objects in context. *European conference on computer vision*. Cham: Springer.

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 3431–3440).

Mohammadi, S., Zuckerman, N., Goldsmith, A., et al. (2016). A critical survey of deconvolution methods for separating cell types in complex tissues. *Proceedings of the IEEE, 105*(2), 340–366.

Noh, H., Hong, S., & Han, B. (2015). Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE international conference on computer vision* (pp. 1520–1528).

Noh, H., Hong, S., & Han, B. (2015). Learning deconvolution network for semantic segmentation. *Proceedings of the IEEE international conference on computer vision*.

Paszke, Adam, et al. Pytorch: An imperative style, high-performance deep learning library. arXiv preprint arXiv:1912.01703 (2019).

Peng, S., et al. (2019). Pvnet: Pixel-wise voting network for 6dof pose estimation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

O. Ronneberger, P. Fischer, and T. Brox, U-net: Convolutional networks for biomedical image segmentation. In; International Conference on Medical image computing and computer-assisted intervention. Springer, 2015, pp. 234–241.

Shen, F., et al. (2017). Semantic segmentation via structured patch prediction, context crf and guidance crf. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Simonyan, Karen, and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014).

Škrabánek, P. (2016). Refined Max-Pooling and Unpooling Layers for Deep Convolutional Neural Networks Mendel 2016. *22nd International Conference on Soft Computing. Vysoké učení technické v Brně*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., et al. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1–9).

Wang, X., Kong, T., Shen, C., et al. (2020). *Solo: Segmenting objects by locations European Conference on Computer Vision* (pp. 649–665). Cham: Springer.

Xie, E., Sun, P., Song, X., et al. (2020). Polarmask: Single shot instance segmentation with polar representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 12193–12202).

Xiong, Y., Liao, R., Zhao, H., et al. (2019). Upsnet: A unified panoptic segmentation network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 8818–8826).

Zagoruyko S, Lerer A, Lin T Y, et al. A multipath network for object detection. arXiv preprint arXiv:1604.02135, 2016.

Zeiler, M. D., et al. (2010). Deconvolutional networks. *2010 IEEE Computer Society Conference on computer vision and pattern recognition*.

Zeiler, M. D., Taylor, G. W., & Fergus, R. (2011). Adaptive deconvolutional networks for mid and high level feature learning. *2011 International Conference on Computer Vision*.

Zhao, H., et al. (2017). Pyramid scene parsing network. *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Pengfei Xian (S'21) received the B.Eng. degree in electrical engineering from Harbin Institute of Technology, Harbin, China, in 2017. He is currently pursuing the Ph.D. degree in electrical engineering at City University of Hong Kong. His research interest includes instance and semantic segmentation on images and videos, deep learning and computer vision.

Lai-Man Po (M'92–SM'09) received the B.S. and Ph.D. degrees in electronic engineering from the City University of Hong Kong, Hong Kong, in 1988 and 1991, respectively. He has been with the Department of Electronic Engineering, City University of Hong Kong, since 1991, where he is currently an Associate Professor of Department of Electrical Engineering. He has authored over 150 technical journal and conference papers. His research interests include image and video coding with an emphasis deep learning based computer vision algorithms.

Dr. Po is a member of the Technical Committee on Multimedia Systems and Applications and the IEEE Circuits and Systems Society. He was the Chairman of the IEEE Signal Processing Hong Kong Chapter in 2012 and 2013. He was an Associate Editor of HKIE Transactions in 2011 to 2013. He also served on the Organizing Committee, of the IEEE International Conference on Acoustics, Speech and Signal Processing in 2003, and the IEEE International Conference on Image Processing in 2010.

Jingjing Xiong received the B.S. degree in Mechanical Design, Manufacturing and Automation from the Xiangtan University, Hunan, China, in 2015, and the M.S. degree in Artificial Intelligence and Pattern Recognition from Shenyang Institute of Automation, Chinese Academy of Sciences, Liaoning, China, in 2018. She is currently pursuing the Ph.D. degree in electrical engineering at City University of Hong Kong, HKSAR, China. Her research interests are in image segmentation, deep learning and computer vision.

Chang Zhou received the BSC degree from the DongHua University of China, Shanghai, in 2016, the Master degree from City University of Hong Kong, Hong Kong, in 2017. He is currently working toward the PhD degree in the Department of Electrical Engineering, City University of Hong Kong. His research interests are in computer vision and deep learning.

Yuzhi Zhao (S'19) received the B.Eng. Degree in electronic information from Huazhong University of Science and Technology, Wuhan, China, in 2018. He is currently pursuing the Ph.D. degree with the Department of Electrical Engineering, City University of Hong Kong. His research interests include image processing, low-level vision and generative model.

Wing-Yin Yu received the B.Eng. degree in Information Engineering from City University of Hong Kong, in 2019. He is currently pursuing the Ph.D. degree at Department of Electrical Engineering at City University of Hong Kong. His research interests are generative adversarial networks, image generation and semantic segmentation.

Wei-Feng Ou received his B.Eng. degree from Guangdong University of Technology in 2013, his M.Eng. degree from South China University of Technology in 2016. He was an engineer in Huawei from 2016 to 2018. He is currently pursuing his Ph.D. degree in City University of Hong Kong. His research interests include deep learning and computer vision.

Yujia Zhang received the B.E. degree in electrical engineering and automation from Huazhong University of Science and Technology in 2015, and the M.S. degree in electrical engineering from South China University of Technology, China, in 2018. He is currently pursuing a Ph. D. degree at City University of Hong Kong. His current research interests include computer vision, video understanding.

Xiaori Zhang received the B.S. degree in Financial Management from Harbin Institute of Technology, China, in 2017. She is currently pursuing the Ph.D. degree at School of Management at Fudan University, China. Her research interests are firm innovation, institutional trading, international market performance and deep learning.