From Pixel to Patch: Synthesize Context-Aware Features for Zero-Shot Semantic Segmentation

Zhangxuan Gu®, Siyuan Zhou, Li Niu®, Zihan Zhao, and Liqing Zhang®, Member, IEEE

Abstract—Zero-shot learning (ZSL) has been actively studied for image classification tasks to relieve the burden of annotating image labels. Interestingly, the semantic segmentation task requires more labor-intensive pixel-wise annotation, but zero-shot semantic segmentation has not attracted extensive research interest. Thus, we focus on zero-shot semantic segmentation that aims to segment unseen objects with only category-level semantic representations provided for unseen categories. In this article, we propose a novel context-aware feature generation network (CaGNet) that can synthesize context-aware pixel-wise visual features for unseen categories based on category-level semantic representations and pixel-wise contextual information. The synthesized features are used to fine-tune the classifier to enable segmenting of unseen objects. Furthermore, we extend pixel-wise feature generation and fine-tuning to patch-wise feature generation and fine-tuning, which additionally considers the interpixel relationship. Experimental results on Pascal-VOC, Pascal-context, and COCO-stuff show that our method significantly outperforms the existing zero-shot semantic segmentation methods.

Index Terms—Contextual information, feature generation, semantic segmentation, zero-shot learning (ZSL).

I. INTRODUCTION

ZERO-SHOT learning [1] aims to deal with the problem of classifying previously unseen categories, and numerous zero-shot classification methods [2]–[12] have been developed. Compared with the popularity of zero-shot classification, zero-shot semantic segmentation has received much less attention. However, semantic segmentation [13]–[22], with the goal to classify each pixel in one image, has much higher annotation cost than the classification task. Thus, it is in high demand to design effective zero-shot semantic segmentation methods.

The setting of zero-shot semantic segmentation [23] is similar to that of zero-shot classification. All categories are divided into seen categories and unseen categories that have no overlap. For ease of description, we refer to the entities (e.g., objects, pixels, features, and word embeddings) belonging to seen/unseen categories as seen/unseen ones. The training images only have pixel-wise annotations for seen categories, while the test images may contain both seen and unseen objects. To enable the trained model to segment unseen objects in the testing stage, we need to transfer knowledge from seen categories to unseen categories via category-level semantic representations (e.g., word embedding such as word2vec [24]).

Among the existing zero-shot segmentation works methods [23], [25]–[31], our work is more related to [23], [26], [30], [31], which are inductive methods aiming to segment multiple categories in an image. The above methods [23], [26], [30], [31] can be divided into two groups: 1) SPNet [26] and Hu et al. [31] learn a mapping from visual features to word embeddings so that classification of all categories can be accomplished in the word embedding space and 2) ZS3Net [23] and CSRL [30] learn a mapping from word embeddings to visual features so that unseen visual features can be generated from unseen word embeddings. Our method belongs to the second group and extends ZS3Net [23], so now we briefly introduce the visual feature generation pipeline used in [23].

Given the word embedding of one category and a random vector, ZS3Net [23] tends to generate pixel-wise visual features for this category in the segmentation network. The generator is trained based on seen categories but expected to produce features for unseen categories. The generated features are used to fine-tune the final classifier to enable segmenting of unseen objects. However, ZS3Net has two severe drawbacks: 1) it simply injects random vectors into the generator to generate diverse features, but the generated features may collapse into a few modes and have limited diversity, which is known as the mode collapse problem [32]–[35] and 2) it only synthesizes pixel-wise visual features, which is limited to fine-tuning the last 1 × 1 convolutional layer.

To overcome the first drawback of ZS3Net, one way to mitigate mode collapse is associating random vectors with specific meaning and establish the one-to-one correspondence between random vector and output [35]. In our task, we conjecture that pixel-wise feature highly depends on its contextual information, in which the contextual information of a pixel means the information inferred from its surrounding pixels (e.g., its location in the object, the pose of the object that it belongs to, and background objects). If so, we can associate random vectors with contextual information and establish the one-to-one correspondence between pixel-wise contextual information and pixel-wise feature. To confirm this conjecture, we perform K-means clustering on the visual
features (output from the ASPP module in Deeplabv2 [15]) of category “cat” and “sheep” on the Pascal-context [36] dataset. From Fig. 1, it can be seen that pixel-wise visual features are influenced by their contextual information in a complicated and interlaced fashion. When \( K = 2 \), the features from the exterior and interior of cat and sheep are clustered into two different groups. When \( K = 5 \), the form of clusters is affected by adjacent or distant background objects. For instance, the red (respectively, blue) cluster of cat features might be related to the blanket (respectively, green plant). Similarly, the red cluster of sheep features might be related to the water. Therefore, we attempt to encode pixel-wise contextual information into contextual latent code to replace random vector, leading to our Context-aware feature Generation Net-work (CaGNet).

Formally, the difference between ZS3Net and our CaGNet in feature generation is shown in Fig. 2. The feature generator \( G \) in ZS3Net takes word embedding \( w \) and random vector following unit Gaussian distribution \( z \sim \mathcal{N}(0, 1) \) as input to generate pixel-wise fake feature \( \tilde{x} \). In contrast, we use contextual module CM to extract pixel-wise contextual information from feature map \( F \), which is encoded into contextual latent code \( z \). It is worth mentioning that our contextual module CM contains a novel context selector, which can automatically decide the suitable scale of context for each pixel. The feature generator \( G \) in our CaGNet takes in word embedding \( w \) and contextual latent code \( z \) to reconstruct real feature \( x \). This establishes the one-to-one correspondence between pixel-wise contextual information and pixel-wise feature, which enforces different contextual latent codes to generate different visual features. We also enforce the distribution of \( z \) to approach \( \mathcal{N}(0, 1) \) to support stochastic sampling. Given one word embedding \( w \), our model can generate visual features of this category in different contexts by sampling \( z \sim \mathcal{N}(0, 1) \) so that the diversity of generated features can be greatly enhanced. After training the feature generator \( G \) based on seen categories, we can generate pixel-wise features for both seen and unseen categories, which are used to fine-tune the last \( 1 \times 1 \) convolutional layer (pixel-wise fine-tuning) to bridge the gap between seen categories and unseen categories.

To overcome the second drawback of ZS3Net, we extend pixel-wise feature generation to patch-wise feature generation, which can be used to fine-tune the classifier including convolutional layers with kernel size larger than 1 (patch-wise fine-tuning). Patch-wise feature is synthesized on the premise of a reasonable category patch, which encodes the semantic layout of a patch. We choose PixelCNN [37] to synthesize category patches because PixelCNN can explicitly model interpixel dependencies. A similar fine-tuning step for the classifier is performed based on the generated patch-wise features. Through patch-wise fine-tuning that considers interpixel relationship, our CaGNet can further improve the segmentation performance.

In summary, compared with ZS3Net, our CaGNet can produce more diverse features with pixel-wise contextual information and support more advanced fine-tuning with patch-wise features. Compared with our preliminary conference version [38], we extend pixel-wise feature generation and fine-tuning to patch-wise feature generation and fine-tuning. Our contributions can be summarized as follows.

1) We use pixel-wise contextual information as guidance to synthesize diverse and context-aware features for zero-shot semantic segmentation.
2) We integrate pixel-wise feature generation and semantic segmentation coherently in a unified network.
3) We extend pixel-wise features to patch-wise features by designing the corresponding patch-wise feature generation and patch-wise feature fine-tuning.
4) Two minor contributions are: 1) contextual module with a novel context selector and 2) modified PixelCNN to generate plausible category patches.
5) Extensive experiments on three benchmark datasets demonstrate the effectiveness of our method.

II. RELATED WORKS

A. Semantic Segmentation

Semantic segmentation is a long-standing and challenging problem in computer vision. Many methods, such as [14]–[18], [21], [22], have made great progress in utilizing rich contextual information because the category predictions of target objects
are often influenced by nearby objects or background scenes. For example, PSPNet [14] and Deeplab [15] designed specialized pooling layers to fuse the contextual information from feature maps of different scales. U-Net [16] and RefineNet [17] focused on designing network architectures that better combine low-level features and high-level ones for ampler contextual cues. The above works motivate us to incorporate contexts into feature generation. However, all the above standard semantic segmentation models require heavy pixel-wise annotations of all categories during training, so they cannot be applied to the zero-shot segmentation task.

B. Weakly Supervised Semantic Segmentation

Because semantic segmentation task heavily relies on dense pixel-wise annotations, there has been an increasing interest in weakly supervised methods, such as image-level [39]–[41], box-level [42], [43], or scribble-level [44] semantic segmentation, to relieve the burden of pixel-wise annotations. In this article, we take a step further and only provide category-level semantic representations for unseen categories, following the zero-shot semantic segmentation setting [23], [26].

C. Zero-Shot Learning

Zero-shot learning (ZSL) was first introduced by [45] in the classification task. ZSL stands for the setting in which all training data are from seen categories, while test data may come from unseen categories. The key of ZSL is to transfer knowledge from seen categories to unseen ones via category-level semantic representations. Many existing ZSL methods [2]–[4], [12], [46]–[49] attempted to learn a mapping between feature space and semantic space. In some cases, we have one sample for each unseen category, which is referred to as one-shot learning. One-shot learning has been studied in many fields like [50]–[54].

Recently, training a conditional generator to synthesize features for unseen categories is a popular approach in zero-shot classification. For example, the method in [55] first generated features using word embeddings and random vectors, which was further improved by later works [5], [6], [56]–[58]. However, these methods generated visual features without involving any contextual information. In contrast, due to the uniqueness of the semantic segmentation task, we utilize contextual information to facilitate feature generation.

D. Generative Adversarial Networks

The generative adversarial network (GAN) [59] usually consists of two models: the generator that is trained to generate new samples and the discriminator that attempts to classify samples as either real or fake. GAN has led to considerable progress in many real-world applications [32], [58], [60]–[62]. To name a few, for unconditional image generation, BigGAN [60] and LS-GAN [62] aim to generate realistic photographs. For conditional image generation, pix2pix [32] is designed for many image-to-image translation tasks. For a cross-modal generation, StackGAN [63] is proposed to generate realistic-looking photographs from textual descriptions of simple objects, such as birds and flowers. GAN has also been widely used in low-level computer vision tasks. For example, for image inpainting, GAIN [64] describes the use of GAN to perform photograph inpainting or hole filling. For super-resolution, SRGAN [61] has been introduced to generate output images with much higher pixel resolution. In ZSL, many researchers utilize GAN to generate features for image classification [5], [55], [58]. Inspired by their success, we propose a GAN with contextual information to synthesize unseen features for zero-shot semantic segmentation.

E. Zero-Shot Semantic Segmentation

To the best of our knowledge, zero-shot segmentation appeared in some prior works [23], [25]–[31]. Among them, transductive methods [27], [29] require test images during training, while this article focuses on inductive zero-shot semantic segmentation without requiring test images during training. The methods [25], [28] only handle binary segmentation tasks, while this article focuses on general semantic segmentation.

The remaining works [23], [26], [30], [31] are closely related to our work and representative of two groups of methods: 1) mapping from visual features to semantic representations and 2) mapping from semantic representations to visual features. In the first group, SPNet [26] transferred knowledge between seen and unseen categories via the semantic projection layer, which projects features to semantic word embeddings. Another method [31] learned the visual-semantic mappings with global information to automatically discriminate between representative samples and noisy ones during training. In the second group, ZSS3Net [23] used semantic word embeddings to generate pixel-wise unseen features, which are utilized to fine-tune a classifier trained on seen categories. Moreover, to capture spatial object arrangement, ZSS3Net is extended to ZSS3Net(GC) by utilizing the graph convolutional network (GCN) [65] to represent each image as an object-relational graph. However, ZSS3Net(GC)
only considers object-level contextual information, and the object-relational graphs involving unseen categories are usually unavailable. More recently, Li et al. [30] proposed a consistent structural relation learning (CSRL) approach to constrain the generation of unseen visual features by exploiting the structural relations between seen and unseen categories. Our method belongs to the second group and holds several advantages: 1) we leverage rich contextual information to guide feature generation and 2) we can generate both pixel-wise features and patch-wise features. Compared with ZS3Net(GC), our method does not require object-relational graphs involving unseen categories.

III. METHODOLOGY

In the zero-shot semantic segmentation task, all categories are split into seen categories $C^s$ and unseen categories $C^u$, where $C^s \cap C^u = \emptyset$. Training images only have pixel-wise annotations of $C^s$, while test images contain objects of $C^s \cup C^u$. To fulfill the knowledge transfer across categories, we resort to category-level word embeddings $\{w_c | c \in C^s \cup C^u\}$, in which $w_c \in \mathbb{R}^d$ is the word embedding of category $c$.

A. Overview

Generally speaking, any segmentation network can be split into a backbone $E$, and a classifier $C$. $E$ extracts the feature map of an input image and $C$ outputs the final segmentation results based on this feature map. Our CaGNet can be built upon an arbitrary segmentation network. In this article, we adopt Deeplabv2 [15] due to its remarkable performance in semantic segmentation. A standard Deeplabv2 network trained on seen categories is unable to segment unseen objects. We attempt to learn a feature generator $G$ based on seen categories, which can synthesize fake features for unseen categories. The classifier in Deeplabv2 is fine-tuned with the synthesized fake features and, thus, able to segment unseen objects.

Fig. 3 shows the overall architecture of our CaGNet. We add our proposed contextual module CM in Fig. 4 between the backbone $E$ and the classifier $C$. CM can produce pixel-wise contextual latent code, which guides the generator $G$ to generate more diverse and context-aware features. The backbone $E$ and the contextual module CM output real features from the input image. The generator $G$ outputs fake features from the contextual latent code and the word embedding. Real features and fake features are then delivered to the classifier $C$ and the discriminator $D$, which could share partial layers to reduce the number of model parameters [66]. Finally, $C$ outputs segmentation results, and $D$ discriminates real features from fake ones. As shown in Fig. 3, the contextual module CM and the classifier $C$ link the segmentation network $\{E, CM, C\}$ and the feature generation network $\{CM, G, D, C\}$.

Next, we will detail our contextual module in Section III-B and pixel-wise (respectively, patch-wise) feature generator in Section III-C (respectively, III-D). For better representation, we use capital letter in bold (e.g., $X$) to denote a map and small letter in bold (e.g., $x_i$) to denote its pixel-wise vectors. Besides, we use upperscript $s$ (respectively, $u$) to represent seen (respectively, unseen) categories. The main symbols used in this article are summarized in Table I.

B. Contextual Module

Pixel-wise contextual information of a pixel is the aggregated information of its neighboring pixels. We design a contextual module CM, which is the core module of our CaGNet, to gather pixel-wise contextual information for each pixel on the output feature map from the segmentation backbone $E$. For the $n$th image, we use $F_n \in \mathbb{R}^{h \times w \times l}$ to denote
the output feature map from $E$. Our contextual module CM is directly applied to $F_n$, as illustrated in Fig. 3.

1) Context Map and Selector: In our CM, we apply several dilated convolutional (conv) layers [67] to $F_n$ to obtain three context maps $\tilde{F}^0_n, \tilde{F}^1_n, \tilde{F}^2_n \in \mathbb{R}^{h \times w \times l}$, which captures the contextual information of three different scales. Besides, we conjecture that the visual features of some pixels may be mainly affected by small-scale contextual information (e.g., the pose or inner parts of objects), while others may be mainly affected by large-scale contextual information (e.g., distant background objects). Hence, we propose a context selector to learn different scale weights for different pixels adaptively. Particularly, we learn a three-channel scale weight map $A_n = [A^0_n, A^1_n, A^2_n] \in \mathbb{R}^{h \times w \times 3}$, where $A^i_n$ contains the weights of the $i$th scale. Then, we duplicate each $A^i_n$ to $\hat{A}^i_n \in \mathbb{R}^{h \times w \times l}$ and obtain the weighted concatenation of context maps $[\tilde{F}^0_n \odot \hat{A}^0_n, \tilde{F}^1_n \odot \hat{A}^1_n, \tilde{F}^2_n \odot \hat{A}^2_n] \in \mathbb{R}^{h \times w \times 3l}$ with $\odot$ indicating elementwise-multiplication. More details can be found in our conference version [38].

2) Contextual Latent Code: As discussed in Section I, contextual latent code, which is assumed to encode pixel-wise contextual information, functions as both the output of contextual module CM and the input of feature generator $G$. Intuitively, given a pixel in a cat near a tree, its contextual latent code may encode its relative location in the cat, its nearby region in the cat, and background objects, such as the tree.

Next, we will describe how to calculate contextual latent code based on context maps. First, we apply a $1 \times 1$ conv layer to $[\tilde{F}^0_n \odot \hat{A}^0_n, \tilde{F}^1_n \odot \hat{A}^1_n, \tilde{F}^2_n \odot \hat{A}^2_n]$ to obtain $\mu_{z_n} \in \mathbb{R}^{h \times w \times l}$ with pixel-wise $\mu_{z_n}$ and $\sigma_{z_n} \in \mathbb{R}^{h \times w \times l}$ with pixel-wise $\sigma_{z_n}$. Following [59], we sample pixel-wise contextual latent code $z_{n,i}$ from Gaussian distribution $\mathcal{N}(\mu_{z_n}, \sigma_{z_n})$ by using $z_{n,i} = \mu_{z_n} + \epsilon \sigma_{z_n} \in \mathbb{R}^l$, where $\epsilon \sim \mathcal{N}(0, 1)$. We also adopt a KL divergence loss to enforce $\mathcal{N}(\mu_{z_n}, \sigma_{z_n})$ to approach $\mathcal{N}(0, 1)$

$$
\mathcal{L}_{KL} = D_{KL}(\mathcal{N}(\mu_{z_n}, \sigma_{z_n}) || \mathcal{N}(0, 1)).
$$

In this way, our contextual latent code supports stochastic sampling to generate visual features by sampling $z_{n,i}$ from $\mathcal{N}(0, 1)$.

The contextual latent code map $Z_n \in \mathbb{R}^{h \times w \times l}$ is acquired by aggregating all $z_{n,i}$ for the $n$th image. The main function of $Z_n$ is to facilitate feature generation, but it can also be utilized to enhance feature map $F_n$. Inspired by [68], we use $Z_n$ as residual attention to obtain a new feature map $X_n = F_n + F_n \odot \phi(Z_n) \in \mathbb{R}^{h \times w \times l}$, where $\phi$ denotes the sigmoid function. Compared with $F_n$, new feature map $X_n$ can bring a little improvement for the segmentation results, as mentioned in [38]. Note that we use $X_n$ instead of $F_n$ as the target real features for feature generation (see Section III-C).

C. Pixel-Wise Feature Generation

In this section, we first introduce how to train feature generator $G$ based on training images. Note that we only utilize the seen pixels while ignoring the unseen pixels in the training images. When processing the $n$th input image $I_n$, our CM has two outputs: the contextual latent code map $Z_n \in \mathbb{R}^{h \times w \times l}$ with pixel-wise $z_{n,i}$ and the real visual feature map $X_n$ with pixel-wise $x_{n,i}$. We follow [23] to use the down-sampled segmentation label map $Y_n \in \mathbb{R}^{h \times w \times (l+c^w)}$ that has the same spatial resolution as $X_n$. In detail, $c^s_{n,i}$ denotes the category label of the $i$th pixel on $X_n, y^s_{n,i}$ (the $i$th pixel-wise vector in $Y_n$) is the one-hot label vector corresponding to $c^s_{n,i}$. The word embedding of the $i$th pixel is $w^s_{n,i} \equiv w_{s, y^s_{n,i}}$ corresponding to its category label $c^s_{n,i}$.

Then, the generator $G$ takes $z_{n,i}$ and $w^s_{n,i}$ as input to synthesize pixel-wise fake feature $\hat{x}^s_{n,i} = G(z_{n,i}, w^s_{n,i})$. As discussed in Section I, we expect to reconstruct pixel-wise real feature $x^s_{n,i}$ based on $z_{n,i}$ and $w^s_{n,i}$ with the assumption of one-to-one correspondence between pixel-wise feature and pixel-wise contextual information for a specific category. Therefore, we apply an L2 reconstruction loss $\mathcal{L}_{REC}$

$$
\mathcal{L}_{REC} = \sum_{n,i} \|x^s_{n,i} - \hat{x}^s_{n,i}\|_2^2.
$$

The real features $x^s_{n,i}$ and fake features $\hat{x}^s_{n,i}$ are fed into classifier $C$ and discriminator $D$. Similar to most segmentation methods [13]–[15], [17], we train the classifier $C$ using cross-entropy classification loss

$$
\mathcal{L}_{CLS} = -\sum_{n,i} y^s_{n,i} \log(C(x^s_{n,i})).
$$

To make the generated fake features indistinguishable from real ones, we adopt the adversarial loss $\mathcal{L}_{ADV}$ following [62]:

$$
\mathcal{L}_{ADV} = \sum_{n,i} \left[ (D(x^s_{n,i}))^2 + (1 - D(\hat{x}^s_{n,i}))^2 \right]
$$

in which the sigmoid function is used to normalize the discriminator scores from $D$ into $[0, 1]$, with 1 indicating real features and 0 indicating fake ones.

Although our generator $G$ is trained on seen categories, it is able to synthesize pixel-wise features for both seen
and unseen categories. By randomly sampling latent code \( z \sim \mathcal{N}(0, 1) \) to replace the contextual latent code \( z_{n,i} \), we can synthesize fake feature \( G(z, w_c) \) for an arbitrary category \( c \in \mathcal{C}_s \cup \mathcal{C}_u \). Intuitively, \( G(z, w_c) \) stands for the pixel-wise feature of category \( c \) in the context specified by \( z \). The synthesized features will be used to fine-tune the classifier \( C \) and enable segmenting unseen objects (see Section III-E).

### D. Patch-Wise Feature Generation

By using synthetic pixel-wise features, we can only fine-tune \( 1 \times 1 \) convolutional (conv) layer without considering the interpixel relationship. If we can generate patch-wise features (e.g., \( 3 \times 3 \) patch), we can fine-tune the classifier formed by conv layers with larger kernel size (e.g., \( 3 \times 3 \) kernel), which can be better adapted to unseen categories by considering an interpixel relationship. In this article, we synthesize small patches (\( 3 \times 3 \)) rather than large patches because generating large category patches is very challenging considering the shapes, poses, or locations of multiple objects, especially for unseen objects.

From a naive point of view, we can randomly stack pixel-wise features to construct patch-wise features. However, the semantic layout, i.e., the categories of pixels within a patch, may be unreasonable. For example, it can be rarely seen in an image that the ground is above the sky. To pursue plausible semantic layout, we need to generate reasonable patch-wise category labels (i.e., category patch) before generating patch-wise features (i.e., feature patch). Among typical generative models, such as GAN [59], VAE [69], and PixelCNN [37], we choose PixelCNN to synthesize category patches because it can explicitly model the interpixel dependencies as we expect. In particular, PixelCNN learns plausible semantic layouts based on word embeddings, and such knowledge can be transferred from seen categories to unseen categories. Then, with the generated category patches containing both seen and unseen categories, we can synthesize feature patches. In this process, we use the same contextual latent code for all pixels in a patch because the contextual information of neighboring pixels in a small patch should be similar. Finally, the synthesized feature patches are used to fine-tune the classifier.

#### 1) Overview of PixelCNN

In this section, we introduce PixelCNN [37] (denoted as \( P \)), a popular autoregressive generative model that can capture the interpixel dependencies. We creatively modify PixelCNN to generate reasonable category patches instead of natural images as it was conventionally developed for. Conventional PixelCNN [37] scans an image row by row and pixel by pixel. For each pixel, it predicts the distribution of its pixel value (RGB values) conditioned on the pixel values of observed pixels. Thus, conventional PixelCNN takes the observed pixel values as input to predict the remaining pixel values.

In our designed PixelCNN, we take the word embeddings of the observed pixels as input to predict the category distributions of the remaining pixels. By virtue of category-level word embeddings, the knowledge of semantic layout can be transferred from seen categories to unseen categories. For ease of representation, we flatten one \( k \times k \) category patch as a sequence \( \mathbf{c} = (c_1, c_2, \ldots, c_k) \), where these \( k^2 \) pixels are taken from the patch row by row. The category distribution of \( t \)th pixel in the category patch depends on the word embeddings of previous observed pixels, i.e., \( p(c_t|w_{c_1}, w_{c_2}, \ldots, w_{c_{t-1}}) \). Following [37], the estimated logarithmic joint distribution of the category patch log \( p(c) \) can be written as the summation of logarithmic conditional distributions over all pixels in the patch

\[
\log p(c) = \sum_{t=1}^{k^2} \log p(c_t|w_{c_1}, w_{c_2}, \ldots, w_{c_{t-1}}). \tag{5}
\]

Both the training stage and the generation stage of PixelCNN are related to (5), which will be detailed later.

#### 2) Training PixelCNN

We aim to maximize (5) when training PixelCNN with seen categories so that the knowledge of semantic layout in the training patches can be learned. To model the conditional probability \( p(c_t|w_{c_1}, w_{c_2}, \ldots, w_{c_{t-1}}) \) in (5), PixelCNN utilizes \( 3 \times 3 \) masked convolutional layers to avoid seeing the future pixels, as shown in Fig. 5. Following [37], we use two types of masks named mask A and mask B. The only difference between them lies in whether the central entry is 0 or 1. Mask A is only applied to the first layer, and mask B is applied to all the subsequent layers. The first layer uses Mask A to exclude the word embedding of the central pixel when predicting the category distribution of this pixel. More details of masked convolutional layers can be found in [37]. By stacking one mask A layer, several mask B layers, and a \( 1 \times 1 \) conv layer as PixelCNN (see Fig. 5), the output category distribution of each pixel will depend on the word embeddings of all previous pixels, which coincides with (5).

To obtain the training patches to train PixelCNN, we slice the label map \( Y^i \) into real category patches \( \mathbf{y}_{n,i} \in \mathbb{R}^{k \times k \times (|C_s|+|C_u|)} \). Note that we discard those patches containing unseen pixels. The corresponding word embedding patches are \( \mathbf{w}_{n,i} \in \mathbb{R}^{k \times k \times d} \). Taking \( \mathbf{w}_{n,i} \) as input, \( P \) outputs the normalized category distribution of each pixel, i.e., \( P(\mathbf{w}_{n,i}) \in \mathbb{R}^{k \times k \times (|C_s|+|C_u|)} \). After flattening \( \mathbf{y}_{n,i} \) and \( P(\mathbf{w}_{n,i}) \) into one-dim vectors, the training loss of PixelCNN can be represented by the cross-entropy classification loss

\[
\mathcal{L}_P = - \sum_{n,i} \mathbf{y}_{n,i} \log (P(\mathbf{w}_{n,i})). \tag{6}
\]

We optimize \( P \) by minimizing \( \mathcal{L}_P \), which is equivalent to maximizing (5). Note that \( P \) is optimized independently of the network modules in Fig. 3.

#### 3) Category Patch Generation by PixelCNN

Based on the trained model \( P \), we can generate category patches containing both seen and unseen categories in the form of \( \mathbf{y}_{n,i}^{\text{gen}} \in \mathbb{R}^{k \times k \times (|C_s|+|C_u|)} \). As shown in Fig. 5, the training process is paralleled, while the generation process is sequential; that being said, we generate each patch row-by-row and pixel-by-pixel in \( k^2 \) steps. In step \( t \in \{1, 2, \ldots, k^2\} \), the trained PixelCNN model \( P \) takes an incomplete word embedding patch filled with the word embeddings of previous \( t-1 \) pixels as input and outputs the normalized category
distribution of the rth pixel, which is denoted as \( p_t = p(c_t|w_0, w_1, \ldots, w_{c_t}) \in \mathbb{R}^{(|C|+|C'|)} \). The category label of the current rth pixel \( c_t^{\text{iw}} \) should be sampled from \( p_t \), i.e., \( c_t^{\text{iw}} \sim p_t \). We use \( y_t^{\text{iw}} \in \mathbb{R}^{(|C|+|C'|)} \) to represent the one-hot label vector of \( c_t^{\text{iw}} \) and fill this \( y_t^{\text{iw}} \) into the rth pixel of the incomplete category patch. On the whole, we repeat the above procedure for \( t = 1, 2, \ldots, k^2 \) until \( t \) reaches \( k^2 \). Finally, we can get a complete category patch by \( w_{i:n} \).

4) Patch-Wise Feature Generation: Based on the synthesized category patches, we can generate feature patches for both seen and unseen categories. We use \( w_{i:n} \in \mathbb{R}^{k \times k \times d} \) to denote the word embedding patch according to an arbitrary category patch by \( w_{i:n} \) containing both seen and unseen categories. As discussed in Section I, we opt for small (e.g., \( 3 \times 3 \)) patches due to the difficulty in synthesizing reasonable large category patches. Suppose that neighboring pixels within a small patch have similar contextual information; we simply use the same contextual latent code \( z \in \mathbb{R}^l \) randomly sampled from \( N(0, 1) \) for \( k^2 \) pixels in the patch. We repeatedly stack \( z \) to construct the contextual latent code patch \( b_z \in \mathbb{R}^{k \times k \times l} \), which is concatenated with \( w_{i:n} \) as the input of \( G \) to generate feature patch \( G(b_z, w_{i:n}) \). Note that \( G \) consists of several \( 1 \times 1 \) conv layers, so each pixel-wise feature in \( G(b_z, w_{i:n}) \) is still generated independently, as in Section III-C. Then, synthetic patch-wise features for both seen and unseen categories are used to fine-tune the classifier (see Section III-E).

E. Optimization

The procedure of optimizing our CaGNet consists of two steps: 1) training and 2) fine-tuning, as illustrated in Fig. 3. In the first step, both pixel-wise and patch-wise methods share the same training pipeline on seen data (red line). In the second step, we will introduce pixel-wise fine-tuning (blue line) and patch-wise fine-tuning (green line) separately. In the training step, we utilize training data of only seen categories to update the whole CaGNet \((E, CM, G, D, C)\). In the fine-tuning step, we utilize the feature generator \( G \) to generate both seen and unseen features, which are used to fine-tune the classifier \( C \), so that CaGNet can generalize to unseen categories. In this step, only \((G, D, C)\) are updated. \((E, CM)\) are frozen since there are no real visual features for gradient backpropagation. Finally, CaGNet only activates \((E, CM, C)\) during the inference, while \((G, D)\) are not used.

1) Training: The objective function of the training step can be formulated by

\[
\min_{G, E, C, CM} \max_D \mathcal{L}_{\text{CLS}} + \mathcal{L}_{\text{ADV}} + \lambda_1 \mathcal{L}_{\text{REC}} + \lambda_2 \mathcal{L}_{\text{KL}}
\]

where all the four loss terms have been discussed in Section III-C, and \( \lambda_1 \) and \( \lambda_2 \) are hyperparameters. The above minimax problem is optimized in an alternating manner. We first maximize \( \mathcal{L}_{\text{ADV}} \) to update \( D \) and then minimize the entire objective function to update \((G, E, C, CM)\).

2) Fine-Tuning: The fine-tuning step can be based on either pixel-wise features or patch-wise ones, which are called pixel-wise fine-tuning and patch-wise fine-tuning respectively.

3) Pixel-Wise Fine-Tuning: For pixel-wise fine-tuning, the classifier \( C \) consists of two \( 1 \times 1 \) convolutional (conv) layers, as shown in Fig. 6, and the discriminator \( D \) shares the first \( 1 \times 1 \) conv layer with \( C \). First, we construct the \( m \)th synthetic label map \( \tilde{Y}_{m,i}^{\text{iw}} \in \mathbb{R}^{h \times w \times (|C|+|C'|)} \) by randomly stacking pixel-wise one-hot label vector \( \tilde{Y}_{m,i}^{\text{iw}} \). As discussed in [38], using approximately the same number of seen pixels and unseen pixels in each label map can generally achieve good performance. The corresponding word embedding map is \( \tilde{W}_{m,i}^{\text{iw}} \in \mathbb{R}^{h \times w \times d} \) with pixel-wise embedding \( \tilde{w}_{m,i}^{\text{iw}} = w_{m,i}^{\text{iw}} \), in which \( \tilde{c}_{m,i}^{\text{iw}} \) is the category label of \( i \)th pixel. Then, we construct latent code map \( \tilde{Z}_{m,i} \) by randomly stacking pixel-wise latent code \( z_{m,i} \), each of which is sampled from \( N(0, 1) \) independently. Utilizing \( \tilde{Z}_{m,i} \) and \( \tilde{w}_{m,i}^{\text{iw}} \) as the input of generator \( G \), we can generate pixel-wise fake features \( \tilde{X}_{m,i}^{\text{iw}} = G(\tilde{Z}_{m,i}, \tilde{w}_{m,i}^{\text{iw}}) \), giving rise to fake feature map \( \tilde{X}_{m,i}^{\text{iw}} \), as shown in Fig. 6. The objective function of pixel-wise fine-tuning can be written as

\[
\min_{G, C} \max_D \tilde{\mathcal{L}}_{\text{CLS}} + \tilde{\mathcal{L}}_{\text{ADV}}
\]
in which classification loss $\tilde{\mathcal{L}}_{\text{CLS}}$ and adversarial loss $\tilde{\mathcal{L}}_{\text{ADV}}$ are obtained by modifying (3) and (4) respectively,

$$\tilde{\mathcal{L}}_{\text{CLS}} = -\sum_{m,i} \tilde{y}_{m,i} \log(C(\tilde{x}_{m,i}^{s,lu}))$$

$$\tilde{\mathcal{L}}_{\text{ADV}} = \sum_{m,i} (1 - D(G(\tilde{z}_{m,i}, \tilde{w}_{m,i}^{s,lu})))^2.$$  

4) Patch-Wise Fine-Tuning: As mentioned in Section III-D, we generate small category patches, resulting in small feature patches. In our implementation, we synthesize $3 \times 3$ feature patches by default, so we will take $k = 3$ as an example to describe patch-wise fine-tuning. In this case, the classifier $C$ consists of a $3 \times 3$ conv layer ($\text{stride} = 1$) and a $1 \times 1$ conv layer, as shown in Fig. 6. The discriminator $D$ shares the first $3 \times 3$ conv layer with $C$.

One issue is that most of the generated category patches in Section III-D contain more than one category. However, in reality, the majority of $3 \times 3$ patches in the training data only have a single category. For ease of description, we name the category patches with only one category as pure patches and the generated patches with multiple categories as mixed patches. We fuse pure patches with mixed patches at the ratio of $\lambda_r : 1$ to better imitate real training data, where $\lambda_r \in \mathcal{R}$ is a hyperparameter.

It is noteworthy that we encounter two main problems when dealing with mixed patches. One problem is that real category patches in the training data usually contain no more than three categories, but the generated mixed patches often contain more than three categories. Another problem is that the generated mixed patches contain very few unseen pixels due to seen bias. To tackle the above two problems, we use a simple yet effective strategy to control the number of categories and simultaneously ensure more unseen pixels in mixed patches. To be specific, we preset the first $\lambda_f$ pixels in each category patch as the same random unseen category $\tilde{C}^u \in \mathcal{C}^u$, where $\lambda_f \in \{0, 1, 2, \ldots, k^2 - 1\}$ is a hyperparameter. Then, PixelCNN will sequentially generate the remaining pixels $\{\lambda_f + 1, \lambda_f + 2, \ldots, k^2\}$ based on $\lambda_f$ observed pixels, which is analogous to category map inpainting. Finally, we filter out the mixed patches with more than three categories.

The procedure of patch-wise fine-tuning is similar to that of pixel-wise fine-tuning. First, we construct the $m$th label map $\tilde{Y}_{m,i}^{s,lu}$ by randomly stacking generated category patches $\tilde{b}_{m,i}^{s,lu} \in \mathbb{R}$. For each category patch $\tilde{b}_{m,i}^{s,lu}$, we can obtain the corresponding word embedding patch $\tilde{w}_{m,i}^{s,lu}$. The associated latent code patch $\tilde{z}_{m,i}$ is obtained by repeatedly stacking the same random latent code $\tilde{z}$. Then, we construct the $m$th word embedding map $\tilde{W}_{m}^{s,lu}$ (respectively, latent code map $\tilde{Z}_{m}$) by randomly stacking $\tilde{w}_{m,i}^{s,lu}$ (respectively, $\tilde{z}_{m,i}$). Based on $\tilde{W}_{m}^{s,lu}$ and $\tilde{Z}_{m}$, we can generate fake feature map $\tilde{X}_{m}^{s,lu}$ with feature patches $\tilde{b}_{m,i}^{s,lu}$. Recall that the feature generator $G$ consists of several $1 \times 1$ conv layers, so each feature patch is actually generated independently, that is, $\tilde{b}_{m,i}^{s,lu} = G(\tilde{z}_{m,i}, \tilde{w}_{m,i}^{s,lu})$.

The objective function of patch-wise fine-tuning has the same form as (8) with slightly different notations. In detail, we replace $\tilde{Y}_{m,i}^{s,lu}$ (respectively, $\tilde{x}_{m,i}^{s,lu}$) in (9) with $\tilde{b}_{m,i}^{s,lu}$ (respectively, $\tilde{b}_{m,i}^{s,lu}$) and replace $\tilde{w}_{m,i}^{s,lu}$ (respectively, $\tilde{z}_{m,i}$) in (10) with $\tilde{w}_{m,i}^{s,lu}$ (respectively, $\tilde{b}_{m,i}^{s,lu}$). The output adversarial loss map and the classification loss map are of the same size as the input feature map $\tilde{X}_{m}^{s,lu}$. Note that, in the $3 \times 3$ conv layer ($\text{stride} = 1$), when a conv kernel overlaps with more than one generated patch, the covered $3 \times 3$ patch may be an unreasonable patch, leading to an invalid loss entry in the loss map. Therefore, we apply a binary loss mask to the output loss map to discard the invalid loss entries. For example, as shown in Fig. 6, a $6 \times 6$ feature map results in a $6 \times 6$ loss map, but only four loss entries are valid.

5) Optimization Strategy: We use ResNet-101 [70] pre-trained on ImageNet [71] to initialize Deeplabv2 backbone $E$. Then, the training step is applied until our model converges. After that, the training step and the fine-tuning step are performed alternatingly every 100 iterations so that the segmentation model can be gradually adapted to unseen categories. If we perform fine-tuning continuously, the model will be biased toward fake features, and the final segmentation quality will be impaired.

IV. EXPERIMENTS

A. Datasets and Basic Settings

Our experiments are conducted on Pascal-VOC 2012 [72], Pascal-context [36], and COCO-stuff [73] datasets: 1) Pascal-VOC 2012 contains 1464 training images with segmentation annotations of 20 object categories; following [23], [26], we adopt additional supervision from semantic boundary annotations [74] for Pascal-VOC; 2) the Pascal-context dataset contains 4998 training images and 5105 validation images of 33 object/stuff categories; and 3) COCO-stuff has 164k images with dense pixel-wise annotations from 182 categories. To follow the standard train/test set split of three datasets. The seen/unseen category split can be found in the conference version [38]. In the training stage, we only use the pixel-wise annotations of seen categories and ignore other pixels.

In terms of word embedding, we concatenate 300-dim word2vec [24] (trained on Google News) and 300-dim fastText [75] (trained on Common Crawl), leading to 600-dim word embedding. If a category name has multiple words,
TABLE II
ZERO-SHOT SEGMENTATION PERFORMANCES ON PASCAL-VOCA, COCO-STUFF, AND PASCAL-CONTEXT. \("pi\) (RESPECTIVELY, \("pa\) ) MEANS PIXEL-WISE (RESPECTIVELY, PATCH-WISE) FINE-TUNING. \("ST\) STANDS FOR SELF-TRAINING. THE BEST RESULTS WITH OR W/O ST ARE DENOTED IN BOLDFACE, RESPECTIVELY.

| Method                | Pascal-VOCA | COCO-stuff | Pascal-Context |
|-----------------------|-------------|------------|----------------|
|                       | Overall      | Seen       | Unseen         |
|                        | hIoU, mIoU, pixel acc. | mIoU, pixel acc. | mIoU, pixel acc. |
| SPNet [26]            | 0.9002       | 0.5467     | 0.7685         | 0.7093       | 0.7583      | 0.9482      | 0.9458      | 0.0001      | 0.0007      | 0.0001 |
| SPNet-c [26]          | 0.2610       | 0.6515     | 0.7755         | 0.7188       | 0.7800      | 0.8877      | 0.8791       | 0.1563      | 0.2955      | 0.2387 |
| ZS3Net [23]           | 0.2874       | 0.6164     | 0.8107         | 0.7349       | 0.7730      | 0.9296      | 0.8772       | 0.1765      | 0.3647      | 0.2468 |
| CSRL [30]             | 0.3712       | 0.6485     | 0.8132         | 0.7512       | 0.7809      | 0.9126      | 0.8845       | 0.2435      | 0.3876      | 0.3431 |
| Hu et al. [31]        | 0.2682       | 0.6210     | 0.7811         | 0.7208       | 0.7771      | 0.8993      | 0.8659       | 0.1621      | 0.2759      | 0.2276 |
| CaGNet(pi) [38]       | 0.3972       | 0.6545     | 0.8068         | 0.7636       | 0.7840      | 0.8950      | 0.8686       | 0.2659      | 0.4297      | 0.3940 |
| CaGNet(pa)            | 0.4326       | 0.6623     | 0.8068         | 0.7643       | 0.7814      | 0.8745      | 0.8621       | 0.2990      | 0.5176      | 0.4710 |
| ZS3Net+ST [23]        | 0.3328       | 0.6302     | 0.8095         | 0.7382       | 0.7802      | 0.9139      | 0.8569       | 0.2115      | 0.3407      | 0.2637 |
| CaGNet(pi)+ST [38]    | 0.4366       | 0.6577     | 0.8164         | 0.7560       | 0.7859      | 0.8704      | 0.8390       | 0.3031      | 0.5835      | 0.5071 |
| CaGNet(pa)+ST         | 0.4528       | 0.6657     | 0.8036         | 0.7650       | 0.7813      | 0.8527      | 0.8394       | 0.3188      | 0.5939      | 0.5417 |

we average the embeddings of multiple words to obtain the word embedding of this category.

The experimental results for seen categories or unseen categories are analyzed based on four evaluation metrics: pixel accuracy, mean accuracy, mean IoU (mIoU), and harmonic IoU (hIoU) following [26], [38]. Among these evaluation metrics, “mIoU” quantizes the overlap between predicted objects and ground-truth ones, which is more reliable than “accuracy” considering the integrity of objects. In zero-shot segmentation, “hIoU” is more valuable than “mIoU” because it can balance all categories and prevent seen categories from dominating the overall results.

B. Implementation Details

We use Deeplabv2 [15] with ResNet-101 as segmentation backbone. The generator G has three 1 × 1 convolutional (conv) layers (512 intermediate dimension, leaky ReLU, and dropout for each layer). For our pixel-wise based method, classifier C and discriminator D have a shared 1 × 1 conv layer [66] and a separate 1 × 1 conv layer. For our patch-wise based method with patch size k × k (k = 3 by default), we change the conv layer shared by C and D to \([k-1]/2\) conv layers (kernel size = 3, stride = 1, and padding = 1). For PixelCNN, we use four masked conv layers (1A + 3B).

In the training stage, the learning rate is initialized as \(2.5e^{-4}\) and divided by 10 whenever the loss stops decreasing. Most experiments have been carried out with batch size 8 on one Tesla V100. The side length of input images is set to 368. We set \(\lambda_1 = 10, \lambda_2 = 100\) in (7) based on the validation set made by splitting out 20% seen categories as validation categories [38]. Specifically, we split out 20% seen categories as the validation categories. Then, we separate the training set into the subtraining set (80%) and the validation set (20%). Note that, in the subtraining set, the pixel-level annotations for validation categories are ignored. We train our model based on the subtraining set and the semantic representations of seen categories. Then, the trained model is applied to the validation set to determine the optimal hyperparameters based on the validation performance. For category patch generation, we set \(\lambda_f = 5, \lambda_r = 4\), which will be further analyzed in Section IV-E.

C. Comparison With State of the Art

We refer to our pixel-wise based method and patch-wise based method as CaGNet(pi) and CaGNet(pa), respectively. We compare with four baselines: SPNet [26], ZS3Net [23], CSRL [30], and Hu et al. [31]. We also compare with
TABLE III
ABLATION STUDIES OF SPECIAL CASES OF PATCH-WISE FINE-TUNING ON PASCAL-VOC

|          | hIoU | mIoU | S-mIoU | U-mIoU |
|----------|------|------|--------|--------|
| pure     | 0.4105 | 0.6421 | 0.7715 | 0.2795 |
| mixed (random) | 0.4027 | 0.6397 | 0.7801 | 0.2713 |
| mixed (constrained random) | 0.4205 | 0.6553 | 0.7743 | 0.2886 |
| mixed (PixelCNN) | 0.4163 | 0.6542 | 0.7812 | 0.2837 |
| mixed (constrained PixelCNN) | 0.4268 | 0.6601 | 0.7756 | 0.2943 |
| Ours: pure + mixed | 0.4326 | 0.6623 | 0.7833 | 0.2988 |

Fig. 7. Performance variance of CaGANNet(pa) when varying $\lambda_f$, $\lambda_r$ on Pascal-VOC. The dashed lines denote the default values used in our article.

SPNet-c, i.e., an extension of SPNet,\(^1\) which deducts the prediction scores of seen categories by a calibration factor. ZS3Net(GC) in [23] is not included for comparison because the object-relational graphs that they used are unavailable in our setting and difficult to acquire in real-world applications. In addition, we employ the self-training (ST) strategy in [23] for both ZS3Net and our CaGANet. To be exact, we use the trained segmentation model to tag unlabeled pixels in training images and then use the new training set to fine-tune our CaGANet. The above two steps are executed iteratively. Note that previous works [23], [26], [30], [31], [38] reported results in considerably different settings (e.g., backbone, word embedding, and category split). We mainly follow the setting in [26] and [38] and compare all methods in the same setting for fairness. We also compared our CaGANet(pi) with ZS3Net [23] in its setting in our conference version [38], so we omit the results here.

Table II summarizes the experiment results of all methods, which shows that our method achieves significant improvements for “unseen” and “overall” evaluation metrics, especially w.r.t. “mIoU,” and “hIoU.” Although our method underperforms SPNet in some “seen” evaluation cases, our method sacrifices some seen pixels for much better overall performance. We also observe that our CaGANet(pa) outperforms CaGANet(pi) w.r.t. “overall” hIoU and “Unseen” mIoU (two most valuable metrics) on all three datasets, especially on the datasets with fewer categories, such as Pascal-VOC. This demonstrates the advantage of patch-wise fine-tuning, which takes interpixel relationship into consideration.

D. Ablation Studies

The results of ablation studies for our CaGANet(pi) have been reported in our conference version [38], so we omit them here due to page limitation.

To verify the performance gain of patch-wise fine-tuning in our CaGANet(pa), we conduct ablation studies on Pascal-VOC. Recall that we use two types of category patches, i.e., pure patches and mixed patches, to synthesize feature patches. Pure patches only contain a single category, while mixed patches are generated by our PixelCNN and contain multiple categories. Instead of PixelCNN, we can also randomly construct mixed patches with multiple categories. In Table III, we compare the segmentation results when fine-tuning the classifier with pure, mixed (random), or mixed (pixelCNN) patches. As mentioned in Section III-E, we impose two constraints on the mixed patches: 1) preset the first $\lambda_f$ pixels and 2) remove the mixed patches with more than three categories. We also compare the mixed patches with or without constraints in Table III.

The first row obtains slight improvement over our pixel-wise based method in Table II, which shows that patch-wise fine-tuning is beneficial even with pure patches. By comparing the third (respectively, fifth) with the second (respectively, fourth) row, we observe that adding constraints can significantly improve the performance because the abovementioned two constraints can improve the authenticity of category patches and the generalization performance on unseen categories. The fifth row achieves the best results among mixed patches and also significantly outperforms pure patches, which demonstrates the advantage of mixed patches generated by PixelCNN under necessary constraints. The last row is our method, which fuses pure patches with mixed patches (constrained PixelCNN) at the ratio of $\lambda_r$:1 for patch-wise fine-tuning. This method achieves the best results because both pure and mixed patches commonly exist in the real training data.

E. Hyperparameter Analyses

We have analyzed $\lambda_1$ and $\lambda_2$ in (7) in the conference version [38], so we omit them here due to page limitation.

In patch-wise fine-tuning, we explore the impact of two hyperparameters $\lambda_f$, $\lambda_r$. We vary $\lambda_f$ (respectively, $\lambda_r$) within the range [0.8] (respectively, [0.5,16]) and report hIoU(%) in Fig. 7. The hIoU increases when $\lambda_f$ increases from 0 to 5, which indicates the benefit of introducing more unseen information. When $\lambda_f$ is larger than 5, the performance decreases due to the lack of patch variety because preset pixels dominate the patch and only leave few pixels for a generation. When $\lambda_r$ is too large, pure patches account for a large proportion, in which case interpixel relationship across different categories is seldom considered. When $\lambda_r$ is too small, mixed patches will overwhelm pure patches, which contradicts the fact that most real category patches in the training data are pure patches. Consequently, it harms the performance of both seen and unseen categories. Based on Fig. 7, the optimal choice is $\lambda_f = 5$ and $\lambda_r = 4$. We also observe that this choice can consistently achieve the best results on all three datasets.

\(^1\)Our reproduced results of SPNet on the Pascal-VOC dataset are obtained using their released model and code with careful tuning but still lower than their reported results.
TABLE IV
PERFORMANCES (HIOU) OF CAIGNET(PA) WITH DIFFERENT PATCH SIZE $k$ ON THREE DATASETS

| patch size | Pascal-VOC | COCO-stuff | Pascal-Context |
|------------|------------|------------|----------------|
| $k = 3$    | 0.4326     | 0.1984     | 0.2135         |
| $k = 5$    | 0.4065     | 0.1851     | 0.2071         |
| $k = 7$    | 0.3767     | 0.1575     | 0.1772         |

Fig. 8. Visualization of zero-shot segmentation results on Pascal-VOC. GT mask is the ground-truth segmentation mask. The categories are marked at the bottom, where red denotes unseen ones.

F. Different Patch Sizes

As discussed in Section III-D, we opt for small patches and set $k = 3$ by default. On all three datasets, Table IV we have a consistent observation that the performance decreases as the patch size $k$ increases. The unsatisfactory performance with large $k$ might be caused by the following two reasons: 1) generating plausible large-scale category patches is very challenging because the shapes and poses of unseen objects are very hard to imagine and 2) the contextual information of pixels within a large patch will be different from each other, so using the same contextual latent code for all pixels in a patch becomes unreasonable.

G. Computation Cost and Processing Time

We report the inference FLOP and FPS of different methods in Table V. We run all methods on pytorch with a single NVIDIA V100. The input image size is $368 \times 368$. Note that, during inference, only backbone $E$, contextual module CM, and classifier $C$ are used. From the table, we can observe that SPNet holds the smallest FLOP, while the FLOPs of other methods increase as the model complexity increases.

Figure 9. Visualization of feature generation quality of CaGNet(pa) on the Pascal-VOC test set. GT mask is the ground-truth segmentation mask. In the third and fourth columns, we show the reconstruction loss maps between generated feature maps and real feature maps (the darker, the better).

TABLE V
FLOP AND FPS OF DIFFERENT METHODS

| Method        | FLOP  | FPS |
|---------------|-------|-----|
| SPNet [26]    | 99.04G| 15.8|
| ZS3Net [23]   | 102.71G| 11.6|
| CSRL [30]     | 101.58G| 12.1|
| Jhu et al. [31]| 99.65G| 12.7|
| CaGNet(pi)    | 102.98G| 10.5|
| CaGNet(pa)    | 103.25G| 10.2|

Fig. 10. Visualization of scale selection in CaGNet(pi) on Pascal-VOC. GT mask is the ground-truth segmentation mask. In scale selection maps, blue, green, and red mean selecting small-, middle-, and large-scale contexts, respectively.

Compared with ZS3Net, our CaGNet has an extra contextual module. Thus, it takes a little bit more time to feedforward an image than ZS3Net. The difference between CaGNet(pi) and CaGNet(pa) is only the kernel size (1 vs. 3) of classifier. As a result, CaGNet(pi) and CaGNet(pa) are very close in terms of efficiency.

H. Diversity of Generated Unseen Features

As claimed in Section I, we attempt to mitigate the mode collapse problem by establishing the one-to-one correspondence between pixel-wise contextual information
and pixel-wise feature. Thus, we use contextual latent code to replace random vectors and employ feature reconstruction loss. To verify this hypothesis, we compare the diversity of unseen features generated by CaGNet(pi) and ZS3Net on Pascal-VOC. We also compare the diversity with real unseen features. For a fair comparison, we sample 1000 random vectors from \( \mathcal{N}(0, 1) \), which are fed into the feature generator of ZS3Net and CaGNet to generate 1000 fake features for each unseen category. We calculate the averaged pairwise Euclidean distance among the features within each unseen category and then calculate the average over all unseen categories. We also sample 1000 real features for each unseen category and calculate the averaged pairwise distance in the same way.

The results are summarized in Table VI. We can observe that the diversity of real unseen features in CaGNet(pi) and ZS3Net is similar. For the generated unseen features, the diversity of CaGNet(pi) is much larger than ZS3Net and closer to real features, which demonstrates that our CaGNet can generate more diverse unseen features and, thus, improve the segmentation performance on unseen categories.

### I. Qualitative Analyses

In this section, we provide some visualization results of our method on Pascal-VOC.

1) Semantic Segmentation: Fig. 8 shows some segmentation results of baselines and our method. “GT” means the ground-truth segmentation mask. Our CaGNet(pi) and CaGNet(pa) perform favorably on unseen objects, e.g., tv monitor (orange), train (light green), sofa (green), potted plant (dark green), and sheep (dark blue), as shown at the bottom of Fig. 8. Moreover, our CaGNet(pa) can segment unseen objects better than CaGNet(pi), e.g., sofa in the third row and tv monitor in the sixth row, showing that patch-wise fine-tuning is helpful.

2) Feature Generation: We evaluate the effectiveness of feature generation of CaGNet(pa) on test images. On the one hand, one test image is fed into the segmentation backbone to obtain the real feature map. On the other hand, the corresponding word embedding and the latent code are fed into the generator to obtain the generated feature map. Then, we calculate the reconstruction loss map based on these two feature maps, as shown in Fig. 9, where smaller loss (darker pixel) implies better generation quality. “With CM” represents our full method (latent code is contextual latent code produced by CM), and “w/o CM” is a controlled experiment (latent code is random vector). We can see that our CM is able to generate better features for seen categories (e.g., “person”) and unseen categories (e.g., “potted plant,” “sofa,” “tv monitor,” and “sheep”) consistently.

3) Context Selector: A context selector is proposed in our contextual module to select the context of suitable scale for each pixel. We calculate the scale weight map \( [A^n_0, A^n_1, A^n_2] \in \mathbb{R}^{h \times w \times 3} \), which contains three scale weights for each pixel in the feature map \( F_n \). \( A^n_0 \) (respectively, \( A^n_1, A^n_2 \)) represents small scale (respectively, middle scale and large scale) with \( 3 \times 3 \) (respectively, \( 7 \times 7 \) and \( 17 \times 17 \)) receptive field size w.r.t. \( F_n \) according to the calculation method in [76]. By choosing the scale with the largest weight for each pixel, a scale selection map can be obtained.

As shown in Fig. 10, we present some images with their scale selection maps obtained by CaGNet(pi). For better visualization, we use three different colors to represent the most suitable scale for each pixel. The small scale, the middle scale and the large scale are, respectively, denoted by blue, green, and red. It can be observed from Fig. 10 that pixels in discriminative local regions (e.g., animal faces and small objects on the table) tend to be affected by a small scale because small-scale contextual information is sufficient for these pixels to reconstruct features. Other pixels prefer middle or large scale since they require contextual information within a larger scale. By discovering the most suitable scale of context for each pixel, the context selector plays a critical role in our contextual module. More analyses of context selector and contextual model CM can be found in [38].

4) Category Patch Generation: To qualitatively verify the effectiveness of category patch generation using PixelCNN, we visualize some \( 3 \times 3 \) mixed patches (constrained PixelCNN) and mixed patches (constrained random). As mentioned in
Section IV-D, the constraints mean that we preset \( \lambda = 5 \) unseen pixels in each patch and filter out the patches containing more than three categories. For each preset unseen category (the first \( \lambda \) pixels in the patch are set as this unseen category), we first generate 100 mixed patches (constrained random) and 100 mixed patches (constrained PixelCNN). Then, we calculate the logarithmic likelihood of these mixed patches according to (5), which reflects the probability of authenticity and reasonableness of each category patch. Finally, for each preset unseen category, we show the category patch with the largest logarithmic likelihood from mixed patches (constrained random) and mixed patches (constrained PixelCNN), respectively, in Fig. 11. We find that our category patch generation method (“constrained PixelCNN”) beats the random method (“constrained random”) in spatial object arrangement and object-level continuities. For spatial object arrangement, in the fifth column of Fig. 11, “tv monitor,” “sofa,” and “cat” are adjacent in the patch generated by our method, which is reasonable and common in reality. However, “tv monitor,” “cow,” and “bird” are adjacent when using the random method, which rarely happens in the real world. For object-level continuities, in the first column of “constrained random,” two “cat” pixels are split by “train.” On the contrary, the pixels of the same category are usually gathered together in our method, which demonstrates better object-level continuities.

V. CONCLUSION

In our proposed CaGNet for zero-shot semantic segmentation, we utilize contextual information to generate diverse and context-aware features by unifying segmentation and feature generation. We have successfully extended the pixel-wise feature generation and fine-tuning to patch-wise ones. The effectiveness of our method has been proven via plenty of qualitative and quantitative experiments. In the future, we will explore how to generate large feature patches with the consideration of reasonable shapes or poses of unseen objects, which may further enhance the performance of zero-shot semantic segmentation.

REFERENCES

[1] C. H. Lampert, H. Nickisch, and S. Harmeling, “Attribute-based classification for zero-shot visual object categorization,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 36, no. 3, pp. 453–465, Mar. 2014.
[2] Y. Yang, Y. Luo, W. Chen, F. Shen, J. Shao, and H. T. Shen, “Zero-shot hashing via transferring supervised knowledge,” in Proc. 24th ACM Int. Conf. Multimedia, Amsterdam, Netherlands, Oct. 2016, pp. 1286–1295.
[3] A. Habibian, T. Mensink, and C. G. M. Snoek, “Composite concept discovery for zero-shot video event detection,” in Proc. Int. Conf. Multimedia Retr., Orlando, FL, USA, Apr. 2014, pp. 17–24.
[4] T. Long, X. Xu, Y. Li, F. Shen, J. Song, and H. T. Shen, “Pseudo transfer with marginalized corrupted attribute for zero-shot learning,” in Proc. 26th ACM Int. Conf. Multimedia, Seoul, South Korea, Oct. 2018, pp. 1802–1810.
[5] J. Li, M. Jing, K. Lu, Z. Ding, L. Zhu, and Z. Huang, “Leveraging the invariant side of generative zero-shot learning,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Long Beach, CA, USA, Jun. 2019, pp. 7402–7411.
[6] D. Mandal et al., “Out-of-distribution detection for generalized zero-shot action recognition,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Long Beach, CA, USA, Jun. 2019, pp. 9985–9993.
[7] H. Zhang, Y. Long, Y. Guan, and L. Shao, “Triple verification network for generalized zero-shot learning,” IEEE Trans. Image Process., vol. 28, no. 1, pp. 506–517, Jan. 2019.
[8] Y. Guo, G. Ding, J. Han, and Y. Gao, “Zero-shot learning with transferred samples,” IEEE Trans. Image Process., vol. 26, no. 7, pp. 3277–3290, Jul. 2017.
[9] S. Rahman, S. Khan, and F. Porikli, “A unified approach for conventional zero-shot, generalized zero-shot, and few-shot learning,” IEEE Trans. Image Process., vol. 27, no. 11, pp. 5652–5667, Nov. 2018.
[10] D. Huynh and E. Elhamifar, “Fine-grained generalized zero-shot learning via dense attribute-based attention,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2020, pp. 4483–4493.
[11] Y. Yu, Z. Ji, J. Han, and Z. Zhang, “Episodic-based prototype generating network for zero-shot learning,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2020, pp. 14035–14044.
[12] Y. Cheng, X. Qiao, X. Wang, and Q. Yu, “Random forest classifier for zero-shot learning based on relative attribute,” IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 5, pp. 1662–1674, May 2018.
[13] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Boston, MA, USA, Jun. 2015, pp. 3431–3440.
[14] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, “Pyramid scene parsing network,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Honolulu, HI, USA, Jul. 2017, pp. 2881–2890.
[15] J.-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,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 40, no. 4, pp. 834–848, Apr. 2017.
[16] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional networks for biomedical image segmentation,” in Proc. Int. Conf. Med. Image Comput. Comput.-Assist. Intervent., Munich, Germany, Oct. 2015, pp. 234–241.
[17] G. Lin, A. Milan, C. Shen, and I. Reid, “RefineNet: Multi-path refinement networks for high-resolution semantic segmentation,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Honolulu, HI, USA, Jul. 2017, pp. 1925–1934.
[18] Z. Zhang, A. Chen, L. Xie, J. Yu, and S. Gao, “Learning semantics-aware distance map with semantics layering network for amodal instance segmentation,” in Proc. 25th ACM Int. Conf. Multimedia, Nice, France, Oct. 2019, pp. 2124–2132.
[19] J. Fu, J. Liu, Y. Wang, J. Zhou, C. Wang, and H. Lu, “Stacked deconvolutional network for semantic segmentation,” IEEE Trans. Image Process., early access, Jan. 25, 2019, doi: 10.1109/TIP.2019.2895460.
[20] H. Bi, L. Xu, X. Cao, Y. Xue, and Z. Xu, “Polarimetric SAR image semantic segmentation with 3D discrete wavelet transform and Markov random field,” IEEE Trans. Image Process., vol. 29, pp. 6601–6614, 2020.
[21] J. Liu, Y. Wang, Y. Li, J. Fu, J. Li, and H. Lu, “Collaborative deconvolutional neural networks for joint depth estimation and semantic segmentation,” IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 11, pp. 5655–5666, Nov. 2018.
[22] J. Fu, J. Liu, J. Jiang, Y. Li, Y. Bao, and H. Lu, “Scene segmentation with dual relation-aware attention network,” IEEE Trans. Neural Netw. Learn. Syst., vol. 32, no. 6, pp. 2547–2560, Jun. 2021.
[23] M. Bucher, T.-H. Yu, M. Cord, and P. Pérez, “Zero-shot semantic segmentation,” in Proc. Adv. Neural Inf. Process. Syst., Vancouver, BC, Canada, May 2019, pp. 468–479.
[24] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” in Proc. Adv. Neural Inf. Process. Syst., Lake Tahoe, NV, USA, Dec. 2013, pp. 1–9.
[25] N. Kato, T. Yamasaki, and K. Aizawa, “Zero-shot semantic segmentation via variational mapping,” in Proc. IEEE/CVF Conf. Int. Conf. Comput. Vis. Workshop (ICCVW), Seoul, South Korea, Oct. 2019, pp. 1363–1370.
[26] Y. Xian, S. Choudhury, Y. He, B. Schiele, and Z. Akata, “Semantic projection network for zero-and few-label semantic segmentation,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Long Beach, CA, USA, Jun. 2019, pp. 8256–8265.
[27] H. Liu, Y. Wang, J. Zhao, G. Yang, and F. Lv, “Learning unbiased zero-shot semantic segmentation networks via transductive transfer,” 2020, arXiv:2007.00515.
[28] S. Yang, Y. Shi, Y. Wang, J. Wang, and Z. Fei, “Attribute driven zero-shot classification and segmentation,” in Proc. IEEE Int. Conf. Multimedia Expo Workshop (ICMEW), San Diego, CA, USA, Jul. 2018, pp. 1–6.
[73] H. Caesar, J. Uijlings, and V. Ferrari, “COCO-stuff: Thing and stuff classes in context,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Salt Lake City, UT, USA, Jun. 2018, pp. 1209–1218.
[74] B. Hariharan, P. Arbeláez, L. Bourdev, S. Maji, and J. Malik, “Semantic contours from inverse detectors,” in Proc. Int. Conf. Comput. Vis., Barcelona, Spain, Nov. 2011, pp. 991–998.
[75] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, “Bag of tricks for efficient text classification,” in Proc. Conf. Eur. Chapter Assoc. Comput. Linguistics, Valencia, Spain, Apr. 2017, pp. 427–431.
[76] W. Luo, Y. Li, R. Urtasun, and R. S. Zemel, “Understanding the effective receptive field in deep convolutional neural networks,” in Proc. Adv. Neural Inf. Process. Syst., Barcelona, Spain, Dec. 2016, pp. 1–9.

Zhangxuan Gu received the B.E. degree in mathematics from Shanghai Jiao Tong University, Shanghai, China, in 2016, where he is currently pursuing the Ph.D. degree with the Computer Science and Engineering Department. His current research interests include machine learning, deep learning, and computer vision.

Siyuan Zhou received the B.E. degree from Shanghai Jiao Tong University, Shanghai, China, in 2020, where he is currently pursuing the Ph.D. degree with the Computer Science and Engineering Department. His current research interests include machine learning, deep learning, and computer vision.

Li Niu received the B.E. degree from the University of Science and Technology of China, Hefei, China, in 2011, and the Ph.D. degree from Nanyang Technological University, Singapore, in 2017. He was a Post-Doctoral Associate with Rice University, Houston, TX, USA. He is currently an Associate Professor with the Computer Science and Engineering Department, Shanghai Jiao Tong University, Shanghai, China. His current research interests include machine learning, deep learning, and computer vision.

Zihan Zhao received the B.S. degree in chemistry from Shanghai Jiao Tong University, Shanghai, China, in 2020, where he is currently pursuing the Ph.D. degree with the Computer Science and Engineering Department. His current research interests include machine learning, deep learning, and natural language processing.

Liqing Zhang (Member, IEEE) received the Ph.D. degree from Zhongshan University, Guangzhou, China, in 1988. He was promoted to a Full Professor position in 1995 with the South China University of Technology. He worked as a Research Scientist with the RIKEN Brain Science Institute, Japan, from 1997 to 2002. Since September 2002, he has been a Professor with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China. He has authored or coauthored more than 250 articles in journals and international conferences. His current research interests cover computational theory for cortical networks, visual cognitive representation and inference, and statistical learning.