How to make a pizza: Learning a compositional layer-based GAN model

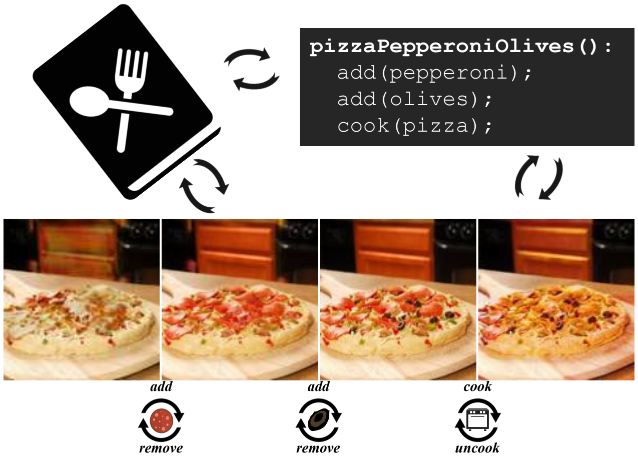
Dim P. Papadopoulos[1] Youssef Tamaazousti[1] Ferda Ofli[2] Ingmar Weber[2] Antonio Torralba[1]
[1] Massachusetts Institute of Technology [2] Qatar Computing Research Institute, HBKU
{dimpapa,ytamaaz,torralba}@mit.edu, {fofli,iweber}@hbku.edu.qa

Abstract

A food recipe is an ordered set of instructions for preparing a particular dish. From a visual perspective, every instruction step can be seen as a way to change the visual appearance of the dish by adding extra objects (e.g., adding an ingredient) or changing the appearance of the existing ones (e.g., cooking the dish). In this paper, we aim to teach a machine how to make a pizza by building a generative model that mirrors this step-by-step procedure. To do so, we learn composable module operations which are able to either add or remove a particular ingredient. Each operator is designed as a Generative Adversarial Network (GAN). Given only weak image-level supervision, the operators are trained to generate a visual layer that needs to be added to or removed from the existing image. The proposed model is able to decompose an image into an ordered sequence of layers by applying sequentially in the right order the corresponding removing modules. Experimental results on synthetic and real pizza images demonstrate that our proposed model is able to: (1) segment pizza toppings in a weakly-supervised fashion, (2) remove them by revealing what is occluded underneath them (i.e., inpainting), and (3) infer the ordering of the toppings without any depth ordering supervision. Code, data, and models are available online[1].

1. Introduction

Food is an integral part of life that has profound implications for aspects ranging from health to culture. In order to teach a machine to "understand" food and its preparation, a natural approach is to teach it the conversion of raw ingredients to a complete dish, following the step-by-step instructions of a recipe. Though progress has been made on the understanding of the recipe-to-image mapping using multi-modal embeddings [6, 39], remaining challenges include (i) the reconstruction of the correct steps in the recipe, and (ii) dealing with partial occlusion of ingredients for food that consists of different layers.



```
pizzaPepperoniOlives():
  add(pepperoni);
  add(olives);
  cook(pizza);
```

Figure 1. **How to make a pizza:** We propose *PizzaGAN*, a compositional layer-based generative model that aims to mirror the step-by-step procedure of making a pizza.

The archetypal example for this is pizza (Fig. 1 (left)). The recipe for making a pizza typically requires sequentially adding several ingredients in a specific order on top of a pizza dough. This ordering of the adding operations defines the overlap relationships between the ingredients. In other words, creating this pizza image requires sequentially rendering different ingredient layers on top of a pizza dough image. Following the reverse procedure of sequentially removing the ingredients in the reverse order corresponds to decomposing a given image into its layer representation (Fig. 1 (right)). Removing an ingredient requires not only detecting all the ingredient instances but also resolving any occlusions with the ingredients underneath by generating the appearance of their invisible parts. Going beyond food, the concept of "layers" is widespread in digital image editing, where images are composed by combining different layers with different alpha masks.

In this paper, we propose *PizzaGAN*, a compositional layer-based generative model that mirrors this step-by-step procedure of making a pizza. Given a set of training images with only image-level labels (e.g., "pepperoni pizza"),

[1] http://pizzagan.csail.mit.edu

for each object class (e.g., "pepperoni"), we learn a pair of module operators that are able to add and remove all instances of the target object class (e.g., "add pepperoni" and "remove pepperoni"). Each such module operator is designed as a generative adversarial network (GAN). Instead of generating a complete new image, each adding GAN module is trained to generate (i) the appearance of the added layer and (ii) a mask that indicates the pixels of the new layer that are visible in the image after adding the layer. Similarly, each removing module is trained to generate (i) the appearance of the occluded area underneath the removed layer and (ii) a mask that indicates the pixels of the removed layer that will not be visible in the image after removing this layer.

Given a test image, the proposed model can detect the object classes appearing in the image (*classification*). Applying the corresponding removing modules sequentially results in decomposing the image into its layers. We perform extensive experiments on both synthetic and real pizzas to demonstrate that our model is able to (1) detect and segment the pizza toppings in a weakly-supervised fashion without any pixel-wise supervision, (2) fill in what has been occluded with what is underneath (i.e., inpainting), and (3) infer the ordering of the toppings without any depth ordering supervision.

2. Related work

Generative Adversarial Networks (GANs). Generative Adversarial Networks (GANs) [2, 11, 16, 36, 38] are generative models that typically try to map an input random noise vector to an output image. GANs consist of two networks, a generator and a discriminator which are trained simultaneously. The generator is trained to generate realistic fake samples while the discriminator is trained to distinguish between real and fake samples. GANs have been used in various important computer vision tasks showing impressive results in image generation [21, 36], image translation [8, 23, 31, 55], high quality face generation [26], super-resolution [29], video generation [12, 44, 45], video translation [3], among many others.

Image-to-image translation. Conditional GANs (cGANs) are able to generate an output image conditioned on an input image. This makes these models suitable for solving image-to-image translation tasks where an image from one specific domain is translated into another domain. Several image-to-image translation approaches based on cGANs have been proposed [5, 8, 22, 23, 31, 34, 47, 55, 56]. Isola et al. [23] proposed a generic image-to-image translation approach using cGANs trained with a set of aligned training images from the two domains. CycleGAN [55] bypasses the need of aligned pairs of training samples by introducing a cycle consistency loss that prevents the two generators from contradicting each other and alleviates the mode collapse problem of GANs.

In this paper, we formulate every object manipulation operator (e.g., add/remove) as an unpaired image-to-image translation and build upon the seminal work of CycleGAN [55]. Our work offers extra elements over the above image-to-image translation approaches by building composable modules that perform different object manipulation operations, generating a layered image representation or predicting the depth ordering of the objects in the image.

Image layers. Decomposing an image into layers is a task that was already addressed in the 90s [10, 20, 42, 43, 46]. More recently, Yang et al. [51] proposed a layered model for object detection and segmentation that estimates depth ordering and labeling of the image pixels. In [50], the authors use the concept of image layers and propose a layered GAN model that learns to generate background and foreground images separately and recursively and then compose them into a final composite image.

Several approaches have been also proposed for the amodal detection [25] or segmentation [15, 30, 57], the task of detecting or segmenting the full extent of an object including any invisible and occluded parts of it. The recent work of Ehsani et al. [13] tries not only to segment invisible object parts but also to reveal their appearance.

Generating residual images. Recently, researchers have explored the idea of using a cGAN model to generate only residual images, i.e., only the part of the image that needs to be changed when it is translated into another domain, for the task of face manipulation [35, 40, 53]. For example, these models are able to learn how to change the hair color, open/close the mouth, or change facial expressions by manipulating only the corresponding parts of the faces. Instead, in this paper, we exploit the generation of residual images to infer a layer representation for an image.

Modular GAN. Our work is also related to studies investigating the modularity and the composability of GANs [17, 53]. Recently, Zhao et al. [53] proposed a modular multi-domain GAN architecture, which consists of several composable modular operations. However, they assume that all the operations are order-invariant which cannot be true for adding and removing overlapping objects in an image. Instead, our model takes into account the layer ordering and is able to infer it at test time without any supervision.

Image inpainting. Removing an object from a natural image requires predicting what lies behind it by painting the corresponding pixels. Image inpainting, the task of reconstructing missing or deteriorated regions of an image, has been widely explored in the past by the graphics community [4, 9, 18]. Recently, several approaches using GANs have been proposed [33, 49, 52] to solve the task. The main difference of our removing modules is that one single GAN

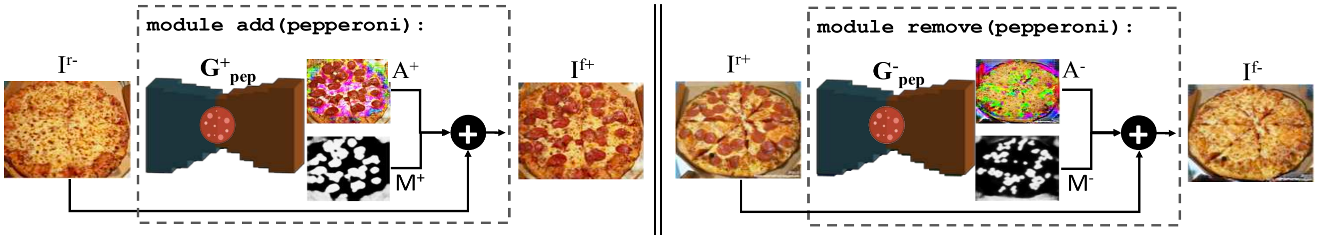


Figure 2. **Module operators that are trained to add and remove pepperoni on a given image.** Each operator is a GAN that generates the appearance A and the mask M of the adding or the removing layer. The generated composite image is synthesized by combining the input image with the generated residual image.

model is responsible for both segmenting the desired objects and generating the pixels beneath them.

3. Method

We now describe our proposed *PizzaGAN* model. In this paper, we are given a set of training RGB images $\mathbf{I} \in \mathbb{R}^{H\times W\times 3}$ of height H and width W with only image-level labels. Let $\mathbf{C} = \{c_1, c_2, ..., c_k\}$ be the set of all the k different labels (i.e., pizza toppings) in our pizza dataset. For each training image I_j, we are given a binary vector of length k that encodes the image-level label (i.e., topping) information for this image.

Our goal is to learn for each object class c two mapping functions to translate images without any instance of class c to images with instances of class c (i.e., *adding* class c) and vice versa (i.e., *removing* class c). To do that, for each class c, we split the training samples into two domains: one with the images which contain the class c (X_c^+) and one with the images that do not contain it (X_c^-).

3.1. Architecture of modules

Generator Module. Let $\mathbf{G}_c^+$ be the generator module that *adds* a layer of the class c on an input image I^{r-} (mapping $\mathbf{G}_c^+ : X_c^- \rightarrow X_c^+$). Also, let $\mathbf{G}_c^-$ be the corresponding generator module that *removes* the layer of the class c (mapping $\mathbf{G}_c^- : X_c^+ \rightarrow X_c^-$). This pair of generator modules is shown in Fig. 2 for the class pepperoni. Below, for simplicity we often omit the class c from the notations.

The output generated images $I^{f+} = \mathbf{G}^+(I^{r-})$ and $I^{f-} = \mathbf{G}^-(I^{r+})$ are given by:

$$I^{f+} = M^+ \odot A^+ + (1 - M^+) \odot I^{r-} \quad (1)$$
$$I^{f-} = M^- \odot A^- + (1 - M^-) \odot I^{r+} \quad (2)$$

where $M^+, M^- \in [0,1]^{H\times W}$ are the layer masks that indicate how each pixel of the adding or the removing layer, respectively, will affect the final composite generated image. $A^+ \in \mathbb{R}^{H\times W\times 3}$ is the RGB image that captures the appearance of the adding layer, while $A^- \in \mathbb{R}^{H\times W\times 3}$ is the RGB image that captures the appearance of the parts that were occluded by the removing layer. In Fig. 2, we observe that A^+ captures the appearance of the pepperoni while A^- captures the appearance of the cheese that lies underneath the pepperoni. The $\odot$ denotes the element-wise product. Note that all the non-zero values of M^+ and M^- denote the pixels that change in the output composite image.

Discriminator. Our model contains a single discriminator $\mathbf{D}$, which is responsible for evaluating the quality of the generated composite images. This network is trained to (i) distinguish whether the input image is real or fake (D_{adv}) and (ii) perform a multi-label classification task for the input image for all the classes (D_{cls}). These two objectives of the discriminator are crucial to force the generator to generate realistic images and, more importantly, to add or remove specific object classes from images without modifying the other class labels of the image. Discriminator networks with an extra auxiliary classification output have been successfully used in various GAN models [8, 32, 35].

3.2. Learning the model

All the adding $\mathbf{G}^+$ and removing $\mathbf{G}^-$ generator modules and the discriminator $\mathbf{D}$ are learned jointly. The full objective loss function contains four different terms: (a) **adversarial** losses that encourage the generated images to look realistic, (b) **classification** losses that prevent the $\mathbf{G}^+$ and $\mathbf{G}^-$ to add or remove instances that belong to a different class than the target one, (c) **cycle consistency** losses that prevent the $\mathbf{G}^+$ and $\mathbf{G}^-$ from contradicting each other, (d) **mask regularization** losses that encourage the model to use both the generated layers and the input image.

Adversarial loss. As in the original GAN [16], we use the adversarial loss to encourage the generated images to look realistic (i.e., match the distribution of the real image samples). For each adding module $\mathbf{G}^+$ and the discriminator $\mathbf{D}$, the adversarial loss is given by:

$$\begin{aligned}\mathcal{L}_{adv}(\mathbf{G}^+, \mathbf{D}) = \mathbb{E}_{I^{r+}}[\log D_{adv}(I^{r+})] + \\ \mathbb{E}_{I^{r-}}[\log(1 - D_{adv}(\mathbf{G}^+(I^{r-})))]\end{aligned} \quad (3)$$

where $\mathbf{G}^+$ aims to generate realistic images, while $\mathbf{D}$ aims to distinguish between real images I^{r+} and fake ones I^{f+}. $\mathbf{G}^+$ tries to minimize this loss, while $\mathbf{D}$ tries to maximize it. Similarly, we introduce an adversarial loss $\mathcal{L}_{adv}(\mathbf{G}^-, \mathbf{D})$ for each removing module $\mathbf{G}^-$ and the discriminator $\mathbf{D}$.

Classification loss. As explained above, the discriminator $\mathbf{D}$ also performs a multi-label classification task. We introduce here a classification loss that encourages the generated

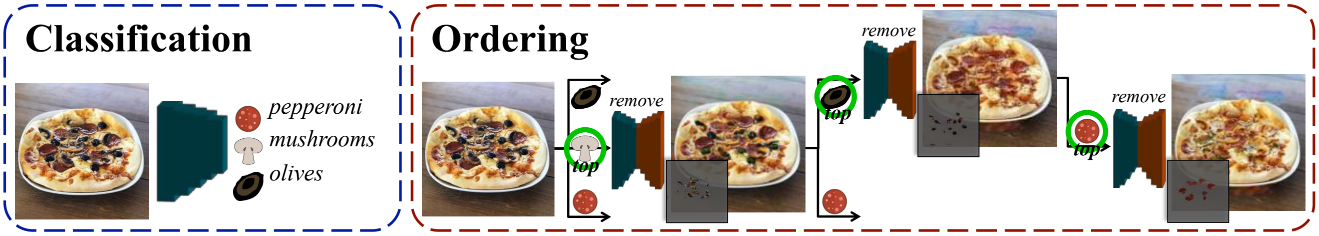


Figure 3. **Test time inference.** Given a test image, our proposed model detects first the toppings appearing in the pizza (classification). Then, we predict the depth order of the toppings as they appear in the input image from top to bottom (ordering). The green circles in the image highlight the predicted top ingredient to remove. Using this ordering, we apply the corresponding modules sequentially in order to reconstruct backwards the step-by-step procedure for making the input pizza.

images to be properly classified to the correct labels. This loss forces the generators to add or remove instances that belong only to the target class while preserving the class labels of all the other classes in the image. Without this loss, certain undesired effects occur such as the removal of class instances that should not be removed or the replacement of instances of an existing class when adding a new one.

This loss consists of two terms: a domain classification loss for the real images that we use to optimize $\mathbf{D}$, and a classification loss for the fake images that we use to optimize $\mathbf{G}^+$ and $\mathbf{G}^-$. More formally we have:

$$\mathcal{L}^{r}_{cls}(\mathbf{D}) = \mathbb{E}_{I^r}[\|D_{cls}(I^r) - l^r\|^2] \tag{4}$$

$$\begin{aligned}\mathcal{L}^{f}_{cls}(\mathbf{G}^+,\mathbf{G}^-,\mathbf{D}) = {} & \mathbb{E}_{I^{r-}}[\left\|D_{cls}(\mathbf{G}^+(I^{r-})) - l^{f+}\right\|^2]+ \\ & \mathbb{E}_{I^{r+}}[\left\|D_{cls}(\mathbf{G}^-(I^{r+})) - l^{f-}\right\|^2]\end{aligned} \tag{5}$$

where D_{cls} represents a probability distribution over all class labels computed by $\mathbf{D}$. l^r represents a vector with the class level information of image I^r while l^{f+} and l^{f-} represent the target class labels of the generated images.

Cycle consistency loss. Using the adversarial and the classification losses above, the generators are trained to generate images that look realistic and are classified to the target set of labels. However, this alone does not guarantee that a generated image will preserve the content of the corresponding input image. Similar to [55], we apply a cycle consistency loss to the generators $\mathbf{G}^+$ and $\mathbf{G}^-$. The idea is that when we add something on an original image and then try to remove it, we should end up reconstructing the original image. More formally, we have:

$$\begin{aligned}\mathcal{L}^{I}_{cyc}(\mathbf{G}^+,\mathbf{G}^-) = {} & \mathbb{E}_{I^{r-}}[\left\|\mathbf{G}^-(\mathbf{G}^+(I^{r-})) - I^{r-}\right\|_1]+ \\ & \mathbb{E}_{I^{r+}}[\left\|\mathbf{G}^+(\mathbf{G}^-(I^{r+})) - I^{r+}\right\|_1]\end{aligned} \tag{6}$$

The cycle consistency loss can be defined not only on the images but also on the generated layer masks. When we first add a layer and then remove it from an input image, the two generated layer masks M^+ and M^- should affect in the same way the exact same pixels of the image. Similar to the above loss, we apply this second consistency loss in both directions:

$$\begin{aligned}\mathcal{L}^{M}_{cyc}(\mathbf{G}^+,\mathbf{G}^-) = {} & \mathbb{E}_{I^{r-}}[\left\|M^+(I^{r-}) - M^-(I^{f+})\right\|_1]+ \\ & \mathbb{E}_{I^{r+}}[\left\|M^-(I^{r+}) - M^+(I^{f-})\right\|_1]\end{aligned} \tag{7}$$

Similar to [55], we adopt the L1 norm for both cycle consistency losses. The final consistency loss L_{cyc} for each pair of $\mathbf{G}^+$ and $\mathbf{G}^-$ is given by the sum of the two terms L^I_{cyc} and L^M_{cyc}.

Mask regularization. The proposed model is trained without any access to pixel-wise supervision, and therefore, we can not apply a loss directly on the generated masks M and the appearance images A. These are learned implicitly by all the other losses that are applied on the final composite generated images. However, we often observe that the masks may converge to zero, meaning the generators have no effect. To prevent this, we apply a regularization loss on the masks M^+ and M^-:

$$\begin{aligned}\mathcal{L}_{reg}(\mathbf{G}^+,\mathbf{G}^-) = {} & \mathbb{E}_{I^{r-}}[\left\|1 - M^+(I^{r-})\right\|_2]+ \\ & \mathbb{E}_{I^{r+}}[\left\|1 - M^-(I^{r+})\right\|_2]\end{aligned} \tag{8}$$

Full loss. The full objective functions for the discriminator $\mathbf{D}$ and for each pair of adding and removing modules (i.e., $\mathbf{G}^+$ and $\mathbf{G}^-$) are defined as:

$$\begin{aligned}\mathcal{L}_D = {} & -\sum_{c=1}^{k}\mathcal{L}_{adv}(\mathbf{G}^+_c,\mathbf{D}) - \sum_{c=1}^{k}\mathcal{L}_{adv}(\mathbf{G}^-_c,\mathbf{D})+ \\ & \lambda_{cls}\sum_{c=1}^{k}\mathcal{L}^r_{cls}(\mathbf{D})\end{aligned} \tag{9}$$

$$\begin{aligned}\mathcal{L}_{G_c} = {} & \mathcal{L}_{adv}(\mathbf{G}^+_c,\mathbf{D}) + \mathcal{L}_{adv}(\mathbf{G}^-_c,\mathbf{D})+ \\ & \lambda_{cls}(\mathcal{L}^f_{cls}(\mathbf{G}^+_c,\mathbf{D}) + \mathcal{L}^f_{cls}(\mathbf{G}^-_c,\mathbf{D}))+ \\ & \lambda_{cyc}(\mathcal{L}_{cyc}(\mathbf{G}^+_c,\mathbf{G}^-_c)) + \lambda_{reg}(\mathcal{L}_{reg}(\mathbf{G}^+_c,\mathbf{G}^-_c))\end{aligned} \tag{10}$$

where λ_{cls}, λ_{cyc} and λ_{reg} are hyper-parameters that control the relative importance of the classification loss, the cycle consistency loss, and the mask regularization loss compared to the adversarial loss.

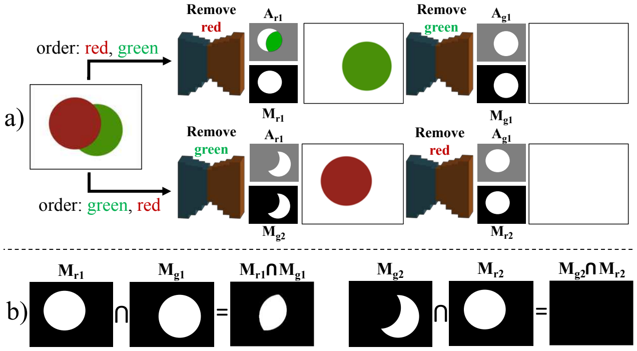


Figure 4. **Predicting the ordering of layers**. (a) A toy example with two overlapping circular objects (red on top of green). In the first row we first remove the red object and then the green one, while in the second row we follow the reverse order. (b) Intersection between the two generated masks for each ordering permutation. We observe that in the first case the two generated masks M highly overlap, while in the second one the overlap is zero.

3.3. Test time inference

At test time, one can arbitrarily stack different composable adding modules and construct a specific sequence of operators. This leads to the generation of a particular sequence of layers which are rendered into an image to create new composite images. This can be seen as an abstraction of generating (making) a pizza image given an ordered set of instructions.

The reverse scenario is to predict the ordered set of instructions that was used to create an image. In other words, given a test image without any supervision, the goal here is to predict the sequence of removing operators that we can apply to the image to decompose it into an ordered sequence of layers. The inference procedure is shown in Fig. 3 and is described below.

Classification. We first feed the image into the discriminator to predict which toppings appear in the image, i.e. which removing modules should be applied.

Ordering. An important question that arises here is which is the right order of applying these module operations to remove the layers. To answer this question we should infer which object is on top of which one. We exploit here the ability of the proposed model to reveal what lies underneath the removed objects. In particular, we use the overlaps of the generate dmasks to infer the ordering of the layers without any supervision.

In Fig. 4, we use a toy example to explain the idea in more details. The image contains two overlapping circular objects with the red circle being on top of the green one. We investigate the two different permutations of ordering (red,green and green,red). We observe that in the first case the two generated masks of the modules highly overlap, while in the second case this overlap is zero (Fig. 4(b)). This happens because the model in the first case reveals green pixels below the red circle (see appearance image A_{r1} in Fig. 4(a)) and completes the occluded green circle. Otherwise the resulting image will contain a green crescent (fake object) and not a green circle (real object). Therefore, we can predict the ordering between two objects by looking which ordering permutation leads to a higher overlap between the generated masks.

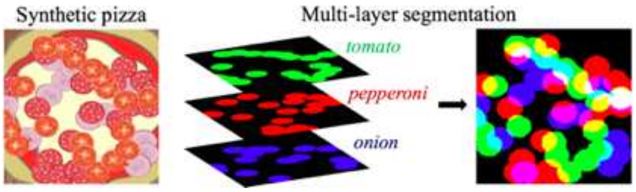


Figure 5. **Ground-truth multi-layer segmentation of a synthetic pizza.** It captures all the occlusions that occur between the different toppings. For example, the cyan pixels (right) denote the parts of the onion occluded by a tomato.

In the general case of predicting the ordering between m different layers, one should ideally try all different $m!$ ordering permutations. This is not feasible in practice. Instead, we can still predict the full ordering by looking solely on the pairwise ordering between the k layers. These results in only $m(m-1)$ pairwise permutations making the ordering inference quite efficient. We also use the difference of the overlaps as an uncertainty measure to resolve contradictions in the pairwise predictions (e.g., a on top of b, b on top of c, c on top of a) by simply ignoring the weakest pairwise ordering prediction (smallest difference of the overlaps).

4. Collecting pizzas

In this section, we describe how we create a synthetic dataset with clip-art-style pizza images (Sec. 4.1) and how we collect and annotate real pizza images on Amazon Mechanical Turk (AMT) (Sec. 4.2).

4.1. Creating synthetic pizzas

There are two main advantages of creating a dataset with synthetic pizzas. First, it allows us to generate an arbitrarily large set of pizza examples with zero human annotation cost. Second and more importantly, we have access to accurate ground-truth ordering information and multi-layer pixel segmentation of the toppings. This allows us to accurately evaluate quantitatively our proposed model on the ordering and the semantic segmentation task. A ground-truth multi-layer segmentation for a synthetic pizza is shown in Fig. 5. Note that in contrast to the standard semantic segmentation, every pixel of the image can take more than one object label (e.g., yellow pixels shown in Fig. 5 (right) denote the presence of both tomato and pepperoni).

We use a variety of different background textures, different clip-art images of plain pizzas, and different clip-art

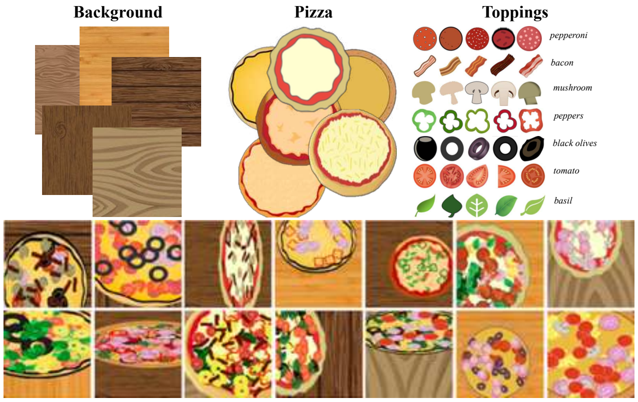


Figure 6. **Creating synthetic pizzas.** Top: Examples of background textures, base pizza images, and toppings used to create synthetic pizzas. Bottom: Examples of created synthetic pizzas.

images for each topping to obtain the synthetic pizzas (examples in Fig. 6 (top)). This adds a more realistic tone to the synthetic dataset and makes the task of adding and removing toppings a bit more challenging. Examples of the obtained synthetic pizzas are shown in Fig. 6 (bottom). The dataset consists of pizzas with a wide variety of different configuration of the toppings (i.e., number of toppings, topping quantities, position of each instance of a topping, and ordering of topping layers).

4.2. Collecting real pizzas

Data. Pizza is the most photographed food on Instagram with over 38 million posts using the hashtag $\#pizza$. First, we download half a million images from Instagram using several popular pizza-related hashtags. Then, we filter out the undesired images using a CNN-based classifier trained on a small set of manually labeled pizza/non-pizza images.

Image-level annotations. We crowd-source image-level labels for the pizza toppings on Amazon Mechanical Turk (AMT). Given a pizza image, the annotators are instructed to label all the toppings that are visible on top of the pizza. Each potential annotator is first asked to complete a qualification test by annotating five simple pizza images. Qualification test is a common approach when crowd-sourcing image annotations as it enhances the quality of the crowd-sourced data by filtering out bad annotators [14, 24, 28, 37, 41].

Annotating pizza toppings can be challenging as several ingredients have similar visual appearances (e.g., bacon-ham, basil-spinach). To further ensure high quality, every image is annotated by five different annotators, and the final image labels are obtained using majority vote.

Data statistics. Our dataset contains 9,213 annotated pizza images and the distribution of the labeled toppings is shown in Fig. 7. The average number of toppings including cheese per pizza is 2.9 with a standard deviation of 1.1.

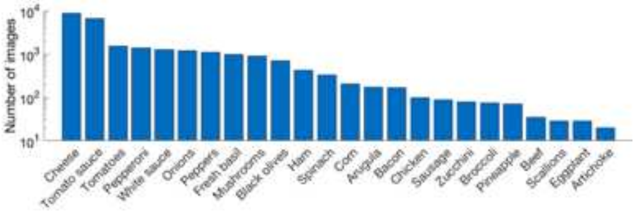


Figure 7. The distribution of the toppings on the real pizzas.

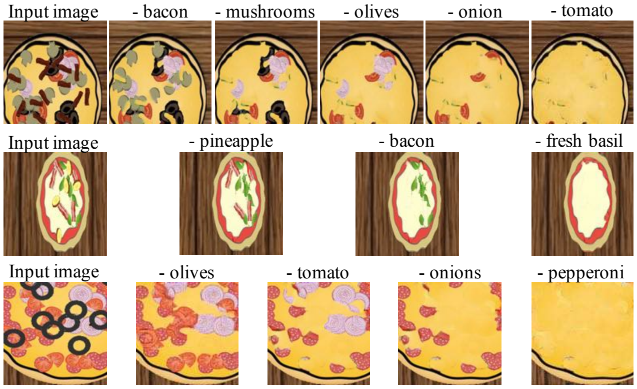


Figure 8. We predict the sequence of removing operators and apply them sequentially to the input image. Note how every time the current top ingredient is the one removed. This process reveals several invisible parts of ingredients when removing the top layers that occlude them.

5. Implementation Details

Architecture. The architectures of the generator modules and the discriminator are based on the ones proposed in [55], as CycleGAN achieves impressive results on the unpaired image-to-image translation. The generator architecture is modified by introducing an extra convolutional layer on top of the second last layer and in parallel with the existing last one. This layer has only one output channel and we use a sigmoid activation function for the mask. For the discriminator, we adopt the popular PatchGAN architecture [23, 55] that we slightly modify to also perform a multi-label classification task.

Training details. We train our model using the Adam solver [27] with a learning rate of 0.0002 for the first 100 epochs. Then, we linearly decay it to zero over the next 100 epochs. All generator modules and the discriminator are trained from scratch. For all the experiments below, we set $\lambda_{cls} = 1$, $\lambda_{cyc} = 10$ and $\lambda_{reg} = 0.01$. For real pizzas, we first get a centered squared crop of the input images and then resize them to 256×256.

6. Experimental results

6.1. Results on synthetic pizzas

Data. We create a dataset of 5,500 synthetic pizzas. Each pizza can contain up to 10 toppings from the following list: $\{$*bacon, basil, broccoli, mushrooms, olives, onions, pepper-*

Method	Architecture	mIoU (%)
CAM [54]	Resnet18 [19]	22.8
CAM [54]	Resnet38+ [48]	39.9
AffinityNet [1]	Resnet38+ [48]	48.2
CAM [54]+CRF	Resnet38+ [48]	51.5
AffinityNet [1]+CRF	Resnet38+ [48]	47.8
PizzaGAN (no ordering)		**56.7**
PizzaGAN (with ordering)		**58.2**

Table 1. **Weakly-supervised segmentation mIoU performance on synthetic pizzas.** All comparison methods are pre-trained on ILSVRC, while PizzaGAN is trained from scratch. In Resnet38+, GAP and FC layers replaced by three atrous convolutions [7].

oni, peppers, pineapple, tomatoes}. We split the images into 5,000 training and 500 test images. We use the training images with accompanying image-level labels to train our model, and measure its performance on the test set.

Qualitative results. Fig. 8 shows qualitative results on synthetic test images. We show how we can predict how a pizza was made: we predict the sequence of removing operators that we can apply to the image to decompose it into an ordered sequence of layers.

Evaluation. Below, we evaluate our model on the following tasks: (i) multi-label topping classification, (ii) layer ordering prediction (see Sec. 3.3), and (iii) weakly-supervised semantic segmentation.

We measure classification performance using mean average precision (mAP). We quantify ordering accuracy using the DamerauLevenshtein distance (DL) as the minimum number of operations (insertion, deletion, substitution, or transposition) required to change the ground-truth ordering into the predicted ordering normalized by the number of ground-truth class labels. We compute segmentation accuracy using the standard Intersection-over-Union (IoU) measure and, then, calculate the mean over all classes (mIoU).

Classification. The classification of the toppings on the synthetic pizzas is a simple task. Our model achieves 99.9% mAP. As a reference, a CNN classifier based on ResNet18 [19] trained from scratch using a binary cross-entropy loss achieves 99.3% mAP.

Ordering. The average normalized DL distance for our PizzaGAN is 0.33. As a reference, a random sequence of random labels achieves 0.91 while a random permutation of the oracle labels achieves 0.42. These numbers express normalized distances, so a lower value indicates higher accuracy. We also evaluate the ordering accuracy only on a subset of the test images that contain exactly two toppings. We find that our method is able to predict the correct ordering 88% of the times.

Segmentation. We compare the segmentation masks generated by the removing modules with various weakly-supervised segmentation approaches (Tab. 1).

Class Activation Maps (CAMs) [54] achieve 22.8% using a ResNet18 architecture. ResNet18 has roughly the same number of parameters with the architecture of our generators. CAMs with a deeper network achieve 39.9%. AffinityNet [1] is a powerful CNN that builds upon CAMs and achieves state-of-the-art results on the PASCAL VOC 2012 dataset. Even though AffinityNet improves CAMs by about 8%, our method outperforms it by $\sim$ 10%. When applying denseCRFs on top of the predicted segments of CAMs and AffinityNet, the performance reaches 51.5% mIoU, which is notably below the performance of our model.

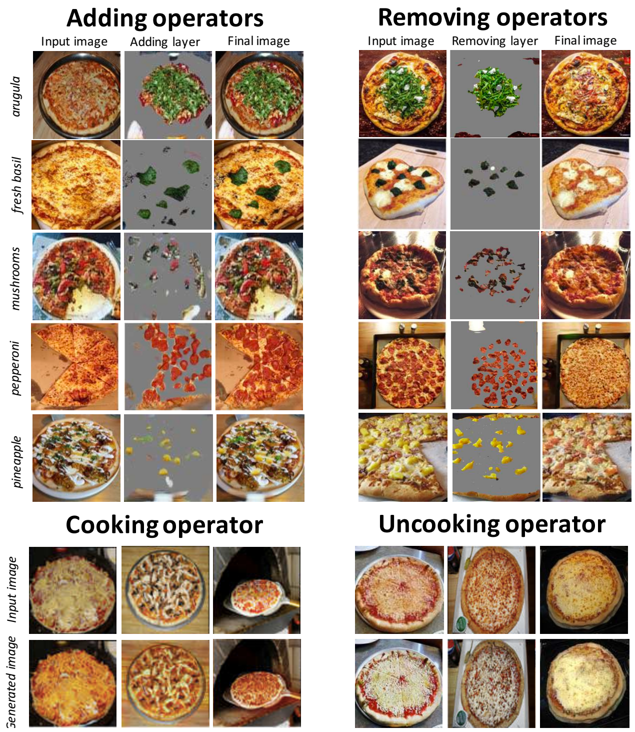


Figure 9. **Qualitative results of individual operators on real pizzas.** (top) Adding and removing operators. (bottom) Cooking and uncooking operators.

Our proposed PizzaGAN without any ordering inference (applying the removing modules in parallel on the input image) achieves 56.7% mIoU. Applying the removing modules sequentially (based on the predicting ordering) brings an additional +2% in mIoU. This reflects the ability of our model to reveal invisible parts of the ingredients by removing the top ones first. Interestingly, using an oracle depth ordering, we achieve 60.9% which is only 3% higher than using the predicted ordering. This upper bound (oracle ordering) provides an alternative way to evaluate the impact of the depth ordering on the segmentation task.

Occluded and non-occluded regions. To further investigate the impact of the ordering, we measure the segmentation performance broken into the occluded and non-occluded regions of the image. Without any ordering prediction, we achieve 70.4% mIoU on the non-occluded re-

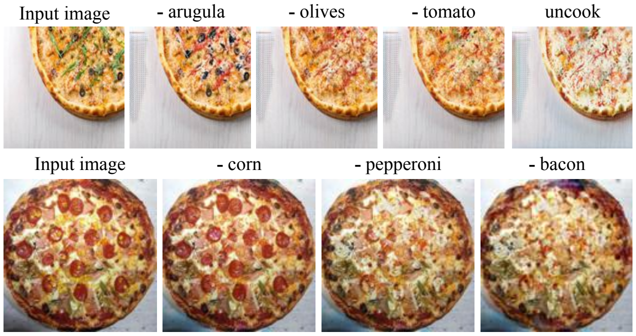


Figure 10. **PizzaGAN**: We predict the sequence of the operaors and apply them sequentially to the input image. The goal of the model is to remove every time the top ingredient. This leads in reconstructing backwards the recipe procedure used to make the input pizza.

gions and 0% mIoU on the occluded ones. Using the predicted depth ordering, we achieve similar performance (70.5%) on the non-occluded regions and 18.2% on the occluded ones. This breakdown shows that the depth ordering enables the prediction of the occluded and invisible parts of the objects which can be very useful for various food recognition applications.

6.2. Experiments on real pizzas

Data. In this section, we perform experiments on real pizzas. We train our model on 9,213 images for 12 classes (toppings): {*arugula, bacon, broccoli, corn, fresh basil, mushrooms, olives, onions, pepperoni, peppers, pineapple, tomatoes*}. For evaluation purposes, we manually annotate a small set of 50 images with accurate segmentation mask (see ground-truth segmentations in Fig. 11). We use the same evaluation setup as for the synthetic pizzas and assess the classification and the weakly-supervised semantic segmentation performance.

Qualitative results. Fig. 9(top) shows the effect of individual adding and removing modules on real images. We observe that the adding modules learn *where* to add by detecting the pizza and *how* to add by placing the new pieces in a uniform and realistic way on the pizza. The removing modules learn *what* to remove by accurately detecting the topping and *how* to remove by trying to predict what lies underneath the removed ingredient. In Fig. 10 we show how we can predict how a pizza was made: we predict the sequence of operators that we can apply to the image to decompose it into an ordered sequence of layers.

Classification. Our model achieves 77.4% mAP. As a reference, a CNN classifier based on ResNet18 [19] trained from scratch using a binary cross-entropy loss achieves 77.6% mAP.

Segmentation. Our approach without any ordering inference (applying the removing modules in parallel on the input image) achieves 28.2% mIoU. Applying the removing modules sequentially using the predicted ordering achieves 29.3% mIoU. As expected the performance is significantly below the one observed on the synthetic data, given that real images are much more challenging than the synthetic ones.

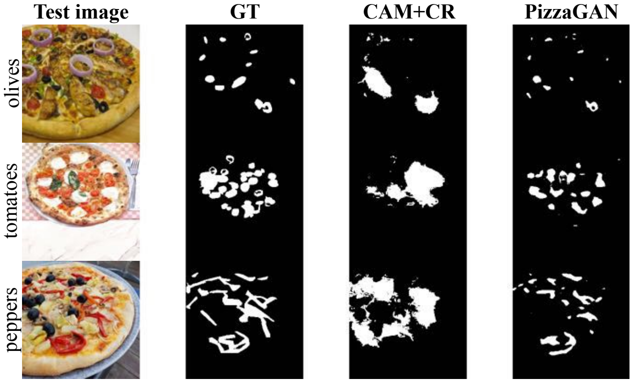


Figure 11. Examples of segmentation results on real pizzas.

Using ResNet38+, CAMs [54] achieve 14.2% and when applying a dense CRF on top of the predictions they achieve 22.7%. Our proposed model outperforms both these models by a large margin (+6.6%). Fig. 11 shows some segmentation prediction examples from CAMs+CRF and our PizzaGAN.

Cooking modules. Besides adding and removing operations, the process of cooking a pizza is essential in the recipe procedure. We manually label here a subset of 932 pizzas as being cooked or uncooked in order to train modules that aim to cook or uncook a given pizza. The modules are trained similarly to the adding/removing ones and some qualitative results are shown in Fig. 9(bottom).

7. Conclusions

In this paper, we proposed *PizzaGAN*, a generative model that mirrors the pizza making procedure. To this end, we learned composable module operations (implemented with GANs) to add/remove a particular ingredient or even cook/uncook the input pizza. In particular, we formulated the layer decomposition problem as several sequential unpaired image-to-image translations. Our experiments on both synthetic and real pizza images showed that our model (1) detects and segments the pizza toppings in a weakly-supervised fashion without any pixel-wise supervision, (2) fills in what has been occluded with what is underneath, and (3) infers the ordering of the toppings without any depth ordering supervision.

Though we have evaluated our model only in the context of pizza, we believe that a similar approach is promising for other types of foods that are naturally layered such as burgers, sandwiches, and salads. Beyond food, it will be interesting to see how our model performs on domains such as digital fashion shopping assistants, where a key operation is the virtual combination of different layers of clothes.

References

[1] Jiwoon Ahn and Suha Kwak. Learning pixel-level semantic affinity with image-level supervision for weakly supervised semantic segmentation. In *CVPR*, 2018. 7

[2] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein GAN. *arXiv*, arXiv preprint arXiv:1701.07875, 2017. 2

[3] Aayush Bansal, Shugao Ma, Deva Ramanan, and Yaser Sheikh. Recycle-gan: Unsupervised video retargeting. In *ECCV*, 2018. 2

[4] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. Image inpainting. In *SIGGRAPH*, 2000. 2

[5] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *CVPR*, 2017. 2

[6] Micael Carvalho, Rémi Cadène, David Picard, Laure Soulier, Nicolas Thome, and Matthieu Cord. Cross-modal retrieval in the cooking context: Learning semantic text-image embeddings. In *SIGIR*, 2018. 1

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

[8] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *CVPR*, 2018. 2, 3

[9] Antonio Criminisi, Patrick Pérez, and Kentaro Toyama. Region filling and object removal by exemplar-based image inpainting. *IEEE Transactions on Image Processing*, 2004. 2

[10] Trevor Darrell and Alexander Pentland. Robust estimation of a multi-layered motion representation. In *Visual Motion, 1991., Proceedings of the IEEE Workshop on*, pages 173–178. IEEE, 1991. 2

[11] E. Denton, S. Chintala, A. Szlam, and Fergus R. Deep generative image models using a laplacian pyramid of adversarial networks. In *NIPS*, 2015. 2

[12] Emily Denton and Rob Fergus. Stochastic video generation with a learned prior. In *ICML*, 2018. 2

[13] Kiana Ehsani, Roozbeh Mottaghi, and Ali Farhadi. SeGAN: Segmenting and generating the invisible. In *CVPR*, 2018. 2

[14] I. Endres and D. Hoiem. Category independent object proposals. In *ECCV*, 2010. 6

[15] Patrick Follmann, Rebecca König, Philipp Härtinger, and Michael Klostermann. Learning to see the invisible: End-to-end trainable amodal instance segmentation. *arXiv preprint arXiv:1804.08864*, 2018. 2

[16] I. Goodfellow, J. Pouget-Abadle, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, 2014. 2, 3

[17] Laura Graesser and Anant Gupta. Composable unpaired image to image translation. *arXiv preprint arXiv:1804.05470*, 2018. 2

[18] J. Hays and A. Efros. Scene completion using millions of photographs. In *SIGGRAPH*, 2007. 2

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 7, 8

[20] Li-wei He, Jonathan Shade, Steven Gortler, and Richard Szeliski. Layered depth images. In *siggraph*, 1998. 2

[21] Xun Huang, Yixuan Li, Omid Poursaeed, John E Hopcroft, and Serge J Belongie. Stacked generative adversarial networks. In *CVPR*, 2017. 2

[22] Xun Huang, Ming-Yu Liu, Serge Belongie, and Jan Kautz. Multimodal unsupervised image-to-image translation. *arXiv preprint arXiv:1804.04732*, 2018. 2

[23] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2016. 2, 6

[24] S. Johnson and M. Everingham. Learning effective human pose estimation from inaccurate annotation. In *CVPR*, 2011. 6

[25] Abhishek Kar, Shubham Tulsiani, Joao Carreira, and Jitendra Malik. Amodal completion and size constancy in natural scenes. In *ICCV*, 2015. 2

[26] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *ICLR*, 2018. 2

[27] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 6

[28] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *ICCV Workshop on 3D Representation and Recognition*, 2013. 6

[29] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew P Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *CVPR*, 2017. 2

[30] Ke Li and Jitendra Malik. Amodal instance segmentation. In *ECCV*, 2016. 2

[31] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks. In *NIPS*, 2017. 2

[32] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. *arXiv preprint arXiv:1610.09585*, 2016. 3

[33] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *CVPR*, 2016. 2

[34] Guim Perarnau, Joost van de Weijer, Bogdan Raducanu, and Jose M Álvarez. Invertible conditional gans for image editing. *arXiv preprint arXiv:1611.06355*, 2016. 2

[35] Albert Pumarola, Antonio Agudo, Aleix M Martinez, Alberto Sanfeliu, and Francesc Moreno-Noguer. Ganimation: Anatomically-aware facial animation from a single image. In *ECCV*, 2018. 2, 3

[36] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016. 2

[37] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. Berg, and L. Fei-Fei. ImageNet large scale visual recognition challenge. *IJCV*, 2015. 6

[38] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training GANs. In *NIPS*, 2016. 2

[39] Amaia Salvador, Nicholas Hynes, Yusuf Aytar, Javier Marin, Ferda Ofli, Ingmar Weber, and Antonio Torralba. Learning cross-modal embeddings for cooking recipes and food images. In *cvpr*, 2017. 1

[40] Wei Shen and Rujie Liu. Learning residual images for face attribute manipulation. In *CVPR*, 2017. 2

[41] Alexander Sorokin and David Forsyth. Utility data annotation with amazon mechanical turk. In *Workshop at CVPR*, 2008. 6

[42] Richard Szeliski, Shai Avidan, and P Anandan. Layer extraction from multiple images containing reflections and transparency. In *cvpr*. IEEE, 2000. 2

[43] Philip HS Torr, Richard Szeliski, and P Anandan. An integrated bayesian approach to layer extraction from image sequences. *IEEE Trans. on PAMI*, 2001. 2

[44] Sergey Tulyakov, Ming-Yu Liu, Xiaodong Yang, and Jan Kautz. Mocogan: Decomposing motion and content for video generation. In *CVPR*, 2018. 2

[45] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Generating videos with scene dynamics. In *NIPS*, 2016. 2

[46] John YA Wang and Edward H Adelson. Representing moving images with layers. *IEEE Transactions on Image Processing*, 1994. 2

[47] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *CVPR*, 2018. 2

[48] Zifeng Wu, Chunhua Shen, and Anton van den Hengel. Wider or deeper: Revisiting the resnet model for visual recognition. *arXiv preprint arXiv:1611.10080*, 2016. 7

[49] Chao Yang, Xin Lu, Zhe Lin, Eli Shechtman, Oliver Wang, and Hao Li. High-resolution image inpainting using multi-scale neural patch synthesis. In *CVPR*, 2017. 2

[50] Jianwei Yang, Anitha Kannan, Dhruv Batra, and Devi Parikh. Lr-gan: Layered recursive generative adversarial networks for image generation. In *ICLR*, 2017. 2

[51] Yi Yang, Sam Hallman, Deva Ramanan, and Charless C Fowlkes. Layered object models for image segmentation. *tpami*, 2012. 2

[52] Raymond A Yeh, Chen Chen, Teck-Yian Lim, Alexander G Schwing, Mark Hasegawa-Johnson, and Minh N Do. Semantic image inpainting with deep generative models. In *CVPR*, 2017. 2

[53] Bo Zhao, Bo Chang, Zequn Jie, and Leonid Sigal. Modular generative adversarial networks. In *ECCV*, 2018. 2

[54] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *CVPR*, 2016. 7, 8

[55] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV*, 2017. 2, 4, 6

[56] Jun-Yan Zhu, Richard Zhang, Deepak Pathak, Trevor Darrell, Alexei A Efros, Oliver Wang, and Eli Shechtman. Toward multimodal image-to-image translation. In *NIPS*, 2017. 2

[57] Yan Zhu, Yuandong Tian, Dimitris N Metaxas, and Piotr Dollár. Semantic amodal segmentation. In *CVPR*, 2017. 2