Learning Program Representations for Food Images and Cooking Recipes

Dim P. Papadopoulos1,3  Enrique Mora2  Nadiia Chepurko1  Kuan Wei Huang1  Ferda Oflı4  Antonio Torralba1

1 MIT CSAIL  2 Nestle  3 DTU Compute  4 Qatar Computing Research Institute, HBKU
dimp@dtu.dk, enrique.mora@es.nestle.com, {nadiia, kwhuang, torralba}@mit.edu, fofli@hbku.edu.qa

Abstract

In this paper, we are interested in modeling a how-to instructional procedure, such as a cooking recipe, with a meaningful and rich high-level representation. Specifically, we propose to represent cooking recipes and food images as cooking programs. Programs provide a structured representation of the task, capturing cooking semantics and sequential relationships of actions in the form of a graph. This allows them to be easily manipulated by users and executed by agents. To this end, we build a model that is trained to learn a joint embedding between recipes and food images via self-supervision and jointly generate a program from this embedding as a sequence. To validate our idea, we crowdsourcing programs for cooking recipes and show that: (a) projecting the image-recipe embeddings into programs leads to better cross-modal retrieval results; (b) generating programs from images leads to better recognition results compared to predicting raw cooking instructions; and (c) we can generate food images by manipulating programs via optimizing the latent code of a GAN. Code, data, and models are available online1.

1. Introduction

Food is an important part of our lives. Imagine an AI agent that can look at a dish and recognize ingredients and reliably reconstruct the exact recipe of the dish, or another agent that can read, interpret and execute a cooking recipe to produce our favorite meal. Computer vision community has long studied image-level food classification [4, 10, 21, 26, 27, 34], and only recently focused on understanding the mapping between recipes and images using multi-modal representations [14, 30, 48, 49, 67]. However, retrieval systems are limited to the existing database and usually fail for queries outside of this database while generating the full recipe from an image remains a challenge [47].

1http://cookingprograms.csail.mit.edu

Cooking recipes are step-by-step instructional procedures which we propose to represent as programs, capturing all the cooking semantics and relationships. A program contains a sequence of actions that can be written as functions (e.g., Cook(), Add()). Each action operates on specific ingredients under certain conditions, such as time or tool (e.g., Cook(pasta, time='10 minutes', tool='pot')). A program also captures the sequential dependency of the actions by maintaining their input-output connections. Note that even though cooking actions are often performed sequentially in time, their connections are not necessarily sequential. The program can also be represented as a graph where each function and parameter is a node, while the edges are the function connections or the connections between the parameters and the actions (Fig. 1).

Our goal is to generate cooking programs conditioned on food images or cooking recipes. We build a model that leverages the natural pairing of food images and recipes by...
3. Cooking programs

In this section, we describe our program representation and show how we design a scalable crowdsourcing protocol to efficiently annotate recipes and obtain cooking programs.

3.1. Program scheme and graph representation

Cooking recipes consist of a title, a list of ingredients, and a list of instruction steps. Each step describes a cooking...
action that operates on ingredients using tools under certain conditions (e.g., time or temperature). This action results in an intermediate output that is used as an input in one of the following steps. This interesting structure, which resembles to the source code of a program, motivates us to use programs to represent cooking recipes (Fig. 1).

A program contains a sequence of functions that correspond to cooking actions. Each function takes as input a list of input variables (ingredients) and parameters (e.g., ingredient quantities or the way the action is performed, such as using a tool). For example, the sentence “Bake the chicken in the oven for 10 minutes at 400 degrees F” can be written as:

\[ h = \text{Bake}(\text{chicken}, \text{tool=}\text{oven}, \text{time=}10 \text{ minutes}, \text{temp=}400 \text{ degrees F}); \]

The output of the function is denoted as \( h \). For a full program, we also capture the input-output connections between the individual commands. Even though a recipe is performed sequentially in time, the output of an action is not always used as an input to the next action. Fig. 1 shows an example where the recipe consists of two sub-recipes that are combined at the end (i.e., cook pasta and prepare the sauce). These connections are captured by using the latent variables \( h \) as inputs to the corresponding functions (e.g.

\[ \text{Drain}(h_1) \]).

Cooking graph. Cooking programs can be also represented as graphs \( G = (V, E) \) (Fig. 1 (right)). Variables and functions are represented as vertices \( V \), while the edges \( E \) are the connections between the functions and their inputs. The graph shows that the sequence of actions can be permuted without violating input-output connections (i.e., the edges \( E \)). We can compute all possible permutations that lead to valid, executable programs, i.e., \( h_i \) cannot be used as input before being computed. The bottom of Fig. 2 shows all valid permutations where the function \( \text{Preheat}() \) can be executed at any point before baking the chicken. In Sec. 4 we use this set of permutations to train our proposed model.

Program taxonomy. Parsing the recipes creates huge vocabularies for each category of the programs (Recipe1M contains 230k unique ingredients, 20k actions and 40k tools). To create a fixed and semantically meaningful vocabulary for each category, we follow a semi-automatic procedure. We describe here the process for the ingredients and we follow a similar one for the others: First, we use Sentence-BERT [45] and extract features for each unique ingredient. Then, we cluster them with K-Means using a high number of clusters (2,000) so that each cluster contains only semantically identical ingredients. Finally, for each cluster, we manually check its nearest neighbor and merge them if they are semantically almost identical. We repeat this iteratively until clusters cannot be merged anymore. Overall, this leads to 514 ingredient, 60 action, 55 tool, 130 quantity, 60 temperature, 152 time, 105 how, 112 why and 220 output clusters.

3.2. Program annotation

Crowdsourcing cooking programs is challenging as most naive annotators have no programming experience. For this reason, we split the process into four simple steps: (a) named-entity annotation, (b) split-and-merge parsing, (c) connection annotation and (d) program taxonomy (Fig. 2).

Named-entity annotation. In this step, we provide annotators a cooking sentence, and ask them to classify (tag) every word as one of the following categories: cooking action, ingredient, quantity, kitchen tool, temperature, time, how, why, and output (Fig. 3 (left)). The annotators first read instructions with the definition of each category and several examples. To ensure high quality responses, we design a protocol that follows common quality control mechanisms [25, 39, 46, 54, 56, 65] and monitor the performance of the annotators by using hidden pre-tagged sentences.

Split and merge. In this step, we perform two parsing operations without any human intervention. First, we split sentences that contain more than one action (e.g., “drizzle with olive oil and bake it for 16-18 minutes” in Fig. 2) so that every sentence contains only one action. Next, we merge tagged words into entities. For example, the phrase “mix fresh oregano, salt and black pepper” has five ingredient words but there are only three ingredients. Black pepper and fresh oregano are merged into the final variables.

Connection annotation. In this step, we provide annotators a recipe with highlighted actions and a panel with these actions as nodes (Fig. 3 (right)), and ask them to connect two...
nodes with an edge if there is an input-output relationship between them. To ensure high quality, we provide instructions and follow standard quality control mechanisms.

**Program taxonomies.** In this step, we follow the variable taxonomies (Sec. 3.1) and for each category (i.e., actions, ingredients, tools, etc.) we map all obtained values of each node into the corresponding categories from our fixed program vocabulary (e.g., “black pepper”→“pepper” in Fig. 2).

**Final program.** Finally, we follow the rules described in Sec. 3.1 to create the final program and the corresponding graph. The output of the last action is denoted as out.

### 3.3. Program collection

We ran experiments on Amazon Mechanical Turk (AMT) and collected programs for 3,708 recipes selected from the Recipe1M dataset [49]. This translates to 42,473 sentences with 478,285 tagged words and 54,154 annotated edge connections. An annotated example is shown in Fig. 2.

**Annotation time and quality.** The median response time of the named-entity task was 17 s per sentence, while the time of the connection task was 75 s per recipe. The annotated recipes have on average 11 sentences leading to a total annotation time of 4 minutes per recipe. The total cost for annotating our programs was about $2,000 (hourly wage about $8). To analyze and quantify the quality of our annotations, we annotate 50 recipes (550 sentences) once more using different annotators and measure the human agreement. The human agreement of the named-entity task is 97.9% (i.e., words with the same annotation label). For the connection task, we found that 7.3% of the connections do not appear in both annotations (agreement of 92.7%).

### 4. From food images and recipes to programs

We introduce the novel task to generate cooking programs from food images or cooking recipes. In Sec. 4.1, we present the model architecture, while in Sec 4.2, we explain the training process and the objective functions.

#### 4.1. Model architecture

Given a set of images \( I \) paired with their recipes \( R \) and a set of programs \( P \), our goal is to learn how to generate a cooking program conditioned on an image or a recipe. Our model consists of three components: (a) a vision encoder, (b) a text encoder, and (c) a program decoder (Fig. 4). The model is trained to embed the images and recipes into the same space via self-supervision and to generate a program from an image or a recipe embedding.

**Vision encoder.** Food images are fed into the vision encoder \( F_V \) based on the Vision Transformer (ViT) [11]. The image is split into fixed-size patches (tokens). After a linear projection and adding position embedding, these tokens are fed to a stack of \( k_v \) Transformer encoder layers [55]. Unlike the original ViT where the image representation is obtained from the “classification token”, we obtain it from the features of all patches after average pooling.

**Text encoder.** The text from cooking recipes is fed into the text encoder \( F_T \) [55]. The architecture of \( F_T \) is similar to \( F_V \) with the difference that here the words play the role of the tokens fed into \( k_t \) identical Transformer layers [55]. Once more, the final recipe representation is obtained after an average pooling layer on top of the token embeddings.
Program decoder. The program decoder $G_P$ consists of $k_p$ transformer decoder layers [55]. The features obtained from $F_V$ are fed into the multi-head attention of each layer following the standard attention mechanism [55]. This results in a predicted program sequence $P_t$ given the image. We repeat the approach for the text part. As such, the features obtained from $F_T$ are fed to $G_P$ to predict a program sequence $P_t$. During inference, a program can be predicted given either a food image or a recipe using the corresponding features and the same decoder $G_P$ (shared weights).

4.2. Training model and loss functions

We train our model end-to-end to jointly learn to project the image features from $F_V$ and the text features from $F_T$ into a common space through a self-supervised loss $L_{ss}$ and to generate programs matching the ground-truth ones from the food images $L_{pv}$ and the cooking recipes $L_{pt}$.

Self-supervised triplet loss. We follow prior work and use a bi-directional max-margin triplet ranking loss to project images and text descriptions into a joint space [18,19,53,59] due to its recent success at the image-to-recipe retrieval task [5,48,57,67]. The loss is based on the triplet ranking loss [51,52] $L_i(a, p, n) = \max(0, s(f(a), f(n)) - s(f(a), f(p)) + m)$ where $a$ is an anchor input, $p$ and $n$ are positive and negative samples, $s$ is a similarity function, $f$ is an embedding and $m$ is a fixed constant margin.

For a mini-batch with size $N$ of image-recipe pairs $\{I_i, R_i\}_{i=1}^N$, we obtain the feature representations of the image $I_i$ and a recipe $R_i$ from the visual $F_V$ and text encoder $F_T$ as $F_V(I_i)$ and $F_T(R_i)$, respectively. The image-recipe pair is considered as positive when $i = j$ and as negative otherwise. The final self-supervised bi-directional loss of the mini-batch, which computes $L_t$ twice, is given by:

$$ L_{ss} = \frac{1}{N} \sum_{i=1, j \neq i}^N \max(0, s(i, j) - s(i, i) + m) + \max(0, s(j, i) - s(i, i) + m) $$

Table 2. Ablation study on ViT feature representation and loss hyperparameters. All models use encoders with 8 layers (small).

Table 1. Cross-modal retrieval results on Recipe1M. At the top part of the table, we report ablation studies of our model, while at the bottom part, we compare our results with the state-of-the-art cross-modal retrieval approaches.

<table>
<thead>
<tr>
<th>Recipe Component</th>
<th>Program Loss</th>
<th>Encoder Layers</th>
<th>image-to-recipe medR</th>
<th>R@1</th>
<th>R@5</th>
<th>R@10 medR</th>
<th>R@1</th>
<th>R@5</th>
<th>R@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>8 (small)</td>
<td>4.6</td>
<td>24.0</td>
<td>54.4</td>
<td>67.8</td>
<td>4.5</td>
<td>23.8</td>
<td>54.2</td>
<td>67.5</td>
</tr>
<tr>
<td>Ingredients</td>
<td>8 (small)</td>
<td>2.8</td>
<td>39.1</td>
<td>68.1</td>
<td>80.3</td>
<td>3.0</td>
<td>38.9</td>
<td>68.4</td>
<td>80.5</td>
</tr>
<tr>
<td>Instructions</td>
<td>8 (small)</td>
<td>3.1</td>
<td>36.6</td>
<td>65.2</td>
<td>76.8</td>
<td>3.0</td>
<td>36.5</td>
<td>65.8</td>
<td>76.9</td>
</tr>
<tr>
<td>Title+Ingr</td>
<td>8 (small)</td>
<td>2.0</td>
<td>43.6</td>
<td>75.6</td>
<td>85.0</td>
<td>2.0</td>
<td>44.9</td>
<td>76.2</td>
<td>85.4</td>
</tr>
<tr>
<td>Title+Ingr+Inst</td>
<td>8 (small)</td>
<td>1.0</td>
<td>53.5</td>
<td>81.8</td>
<td>89.2</td>
<td>1.0</td>
<td>53.1</td>
<td>82.0</td>
<td>89.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State-of-the-art cross-modal retrieval</th>
<th>Img Encoder</th>
<th>image-to-recipe medR</th>
<th>R@1</th>
<th>R@5</th>
<th>R@10 medR</th>
<th>R@1</th>
<th>R@5</th>
<th>R@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Salvador CVPR 17 [49]</td>
<td>ResNet-50</td>
<td>5.2</td>
<td>24.0</td>
<td>51.0</td>
<td>65.0</td>
<td>5.1</td>
<td>25.0</td>
<td>52.0</td>
</tr>
<tr>
<td>Chen ACM MM 18 [8]</td>
<td>ResNet-50</td>
<td>4.6</td>
<td>25.6</td>
<td>53.7</td>
<td>66.9</td>
<td>4.6</td>
<td>25.7</td>
<td>53.9</td>
</tr>
<tr>
<td>Carvalho SIGIR 18 [5]</td>
<td>ResNet-50</td>
<td>2.0</td>
<td>39.8</td>
<td>69.0</td>
<td>77.4</td>
<td>1.0</td>
<td>40.2</td>
<td>68.1</td>
</tr>
<tr>
<td>Zhu CVPR 19 [67]</td>
<td>ResNet-50</td>
<td>2.0</td>
<td>39.1</td>
<td>71.0</td>
<td>81.7</td>
<td>2.0</td>
<td>40.6</td>
<td>72.6</td>
</tr>
<tr>
<td>Fu CVPR 20 [14]</td>
<td>ResNet-50</td>
<td>2.0</td>
<td>48.2</td>
<td>75.8</td>
<td>83.6</td>
<td>1.9</td>
<td>48.4</td>
<td>76.1</td>
</tr>
<tr>
<td>Pham AAAI 21 [42]</td>
<td>ResNet-50</td>
<td>1.6</td>
<td>49.7</td>
<td>79.3</td>
<td>86.3</td>
<td>1.6</td>
<td>50.1</td>
<td>79.0</td>
</tr>
<tr>
<td>Wang CVPR 19 [57]</td>
<td>ResNet-50</td>
<td>1.0</td>
<td>51.8</td>
<td>80.2</td>
<td>87.5</td>
<td>1.0</td>
<td>52.8</td>
<td>80.2</td>
</tr>
<tr>
<td>Wang TMM 21 [58]</td>
<td>ResNet-50</td>
<td>1.0</td>
<td>54.0</td>
<td>81.7</td>
<td>88.8</td>
<td>1.0</td>
<td>54.9</td>
<td>81.9</td>
</tr>
<tr>
<td>Fain arXiv 19 [13]</td>
<td>ResNet-101</td>
<td>1.0</td>
<td>60.2</td>
<td>84.0</td>
<td>89.7</td>
<td>1.0</td>
<td>60.3</td>
<td>87.6</td>
</tr>
<tr>
<td>Salvador CVPR 21 [48]</td>
<td>ResNet-50</td>
<td>1.0</td>
<td>60.0</td>
<td>87.6</td>
<td>92.9</td>
<td>1.0</td>
<td>60.3</td>
<td>87.6</td>
</tr>
<tr>
<td>Salvador CVPR 21 [48]</td>
<td>ViT-B/16</td>
<td>1.0</td>
<td>63.2</td>
<td>88.3</td>
<td>93.1</td>
<td>1.0</td>
<td>66.8</td>
<td>89.8</td>
</tr>
</tbody>
</table>

| Ours                                  | ViT-B/16    | 1.0                  | 66.9| 90.9| 95.1    | 1.0 | 66.8| 94.6 |

$\lambda_{pv} = 0.1, \lambda_{pt} = 0.1$ average 58.6 85.7 91.7

$\lambda_{pv} = 0.1, \lambda_{pt} = 0.1$ average 56.1 83.8 90.1

$\lambda_{pv} = 0,\lambda_{pt} = 0$ average 53.5 81.8 89.2

$\lambda_{pv} = 1, \lambda_{pt} = 1$ average 58.0 85.1 90.9

$\lambda_{pv} = 10, \lambda_{pt} = 10$ average 56.4 83.7 89.6
Program prediction loss. Each recipe $R_i$ is a sequence of $K$ sentences $(r^1, r^2, ..., r^K)$. The program $P_k$ is a sequence of $L$ programming function commands $(p^1, p^2, ..., p^L)$ following the ordering of the recipe instructions (Fig. 1). However, as described in Sec. 3.1, some of the commands $p^k$ can be permuted without violating the input-output connections between the functions. We denote as $P_i = (P_1^i, P_2^i, ..., P_k^i)$ the set of all valid permutations of the sequence $P_i$. At each time step, $G_p$ predicts a probability distribution for the output tokens, which is typically followed by a softmax and a cross-entropy loss between the predicted and the target output tokens, which is typically followed by a softmax and a cross-entropy loss between the predicted and the target sentence. We denote as $\Pi_i = (\Pi_1^i, \Pi_2^i, ..., \Pi_k^i)$ the generated program condition on $R_i$.

Let $\Pi_i^k = G_p(F_V(I_i))$ be the generated program condition on $I_i$ and $\Pi_i^k = G_p(F_T(R_i))$ be the generated program condition on $R_i$. The loss $L_{pt}$ for a mini-batch is given by:

$$L_{pt} = \frac{1}{N} \sum_{i=1}^{N} \min_{j \in [1, \beta]} L_{ce}(\Pi_i^j, P_i^j)$$

(2)

where $L_{ce}(\Pi_i^j, P_i^j)$ is the cross-entropy loss between $\Pi_i$ and the $j^{th}$ target program sequence $P_i^j$. $L_{pt}$ is given by eq. (2) by replacing $\Pi_i^k$ with $\Pi_i^k$.

Final loss. The full objective function $L$ of our model is defined as $L = \lambda_{ss} L_{ss} + \lambda_{pt} L_{pt} + \lambda_{pv} L_{pv}$, where $\lambda_{ss}$, $\lambda_{pt}$, $\lambda_{pv}$ are hyperparameters that control the relative importance of the image-recipe loss to the program prediction losses.

5. Experimental results

This section presents our experimental results. We evaluate our approach on three tasks: image-to-recipe retrieval (Sec. 5.1), program generation from recipes and images (Sec. 5.2), and image generation from programs (Sec. 5.3).

Data. We use the Recipe1M dataset [49] as standard in previous work [13,14,48,49,57,58,67]. Recipe1M contains 887,706 food images and 1,029,720 cooking recipes split in training (70%), validation (15%) and test (15%) sets. Note

\[ s(i, j) = s(F_V(I_i), F_T(R_j)) \] denotes the cosine similarity between image $I_i$ and a recipe $R_j$.

That we only use the recipes that have corresponding images (340,831 recipes). For the cooking programs, we use our ground-truth dataset with 3,708 programs.

Implementation details. Unless stated otherwise, we use the following settings. $F_V$ is based on ViT-B/16 [11] ($k_v=12$ layers, 12 heads) originally pretrained on ImageNet [46]. $F_T$ and $G_P$ are based on [55]. We use $k_t=8$ encoder layers (8 heads) and 2 decoder layers (4 heads).

To handle the imbalance between the image-recept and the recipe-program pairs, we first pre-train $F_T$ and $G_P$ alone using our annotated recipe-program pairs and obtain pseudo-ground-truth programs for the whole Recipe1M. The images are resized to $256 \times 256$ and then cropped to $224 \times 224$. During training we perform random crop and horizontal flip augmentation. We train our model for 50 epochs using the Adam optimizer [23] with a base learning rate of $10^{-3}$ and
a step decay of 0.1 every 20 epochs. We set $\lambda_{ss} = 1$, $\lambda_{pv} = 0.1$, $\lambda_{pt} = 0.1$ and $m = 0.3$. All experiments were run on four Nvidia Titan X GPUs.

### 5.1. Image-to-recipe retrieval

**Evaluation.** Following the protocol of Recipe1M [49], we measure the retrieval performance on the test set with median rank (medR) and recall at top K (R@K) for $K = 1, 5, 10$ on ranking of 1,000 recipe-image pairs. We report the average metrics after repeating experiments 10 times.

**Recipe components.** We first examine the effect of training and testing the model using different recipe components. In the first three rows of Tab. 1, we observe that the ingredient list is the most informative component (R@1=39.1). Combining it with the title gives a small boost in performance (+4.5 in R@1), while using the full recipe yields 53.5.

**Predicting programs.** The first five models were trained without the decoder $G_P$. We observe that training the model using $G_P$ to predict programs from the common embedding space significantly improves the performance (+4.9 R@1).

**Deeper encoders.** When using deeper encoders (Base model with 12 layers), the R@1 accuracy jumps to 66.9.

**Comparison with state of the art.** In Tab. 1(bottom), we compare our final model against several approaches. We achieve state-of-the-art performance thanks to the powerful transformer encoders and our program prediction loss. Note that a fair comparison among all methods is hard since they all use different image (ResNet, ResNext, ViT) and text (LSTMs, transformers) encoders. Also, note that previous work [48] use multiple transformer encoders for each component of the recipe. Instead, we use only a single text encoder (Fig. 4) leading to a much simpler and efficient model. Also, we do not use any unpaired data from Recipe1M unlike most of the existing approaches [5,13,48,49,57,67].

**ViT features.** Using features from the “cls token” as in the original ViT [11] (instead of average pooling) causes a drop in performance (-2.5 in R@1), as shown in Tab. 2(top).

**Loss hyperparameters.** In Tab. 2, we perform an ablation study for $\lambda_{pv}$ and $\lambda_{pt}$ showing a small performance drop (-0.6% with $\lambda_{pv} = 1$ and -2.2% with $\lambda_{pv} = 10$ in R@1).

### 5.2. Program prediction from images and recipes

We evaluate here our model on the task of predicting programs from images or recipes (Fig. 5, Tab. 3).

**Evaluation.** Instead of considering the program as a sequence and use standard sequence evaluation metrics [40], we turn the programs into graphs and measure the graph edit distance (GED) [50] wrt to the groundtruth one. GED captures not only the semantic nodes but also their ordering and topology. This evaluation is feasible due to our fixed program vocabulary. Moreover, we extract only the ingredient nodes of the graphs and measure accuracy with the F1 score between the predicted and the ground-truth sets. We follow the same strategy for the action and tool nodes.

**Programs from images.** In Tab. 3(top), we report our results for predicting programs from images and compare with several baselines: (a) Random image: program from a random image; (b) Retrieved image: program from the top retrieved image using our retrieval system; (c) NN (oracle): oracle nearest neighbor (NN) program (e) Instructions: our model where we decode the instruction text instead of a program. Tab. 3(top) shows that generating programs leads to better results (better ingredient, action and tool prediction) than the strong baselines of decoding sentences or relying on a retrieval system for making predictions. Interestingly, our model is only slightly below the oracle baseline that simulates an ideal retrieval system. We also observe that our program loss (minCE) slightly improves results over the standard CE loss. Fig. 6 show qualitative results for the ingredient prediction on food images, while Fig. 5 shows a
predicted program from a recipe and an image.

**Programs from recipes.** Similarly, in Tab. 3(bottom), we report our results for predicting programs from recipes and repeat the above baselines. The text classifier here is a transformer-based multi-label classifier that predicts ingredients. As expected, predicting programs from recipes is a much simpler task than predicting them from images. Once more, we observe similar trends and our loss yields slightly better results than the standard CE and significantly better than the Instructions baseline (+33.9% F1 for ingredients).

**Attention.** To further understand how $F_v$ and $F_T$ process images and text, we visualize their attention weights following [1]. In Fig. 7 we observe that both encoders focus on semantically similar concepts across the two domains.

### 5.3. Image generation

We show an interesting application of image generation conditioned on cooking programs using our model. We first train a StyleGANv2 [20] on the images of the Recipe1M dataset [49]. Given an initial latent vector $z_0$, we use the GAN to generate an image. The image goes through the encoder $F_v$ and the decoder $G_P$ to obtain a program. We compute the loss between this program and the desired input one. We use the loss to optimize $z$ via backpropagation. Similar approaches have been proposed using the recently introduced CLIP [44] model to drive image generation using text sentences [3, 41]. Fig. 8(left) illustrates our approach while in Fig. 8(right) we show examples of generated images using a full program or a part of it (green nodes only). We observe that the generated images are realistic and capture plausibly the content of the input program.

### 6. Conclusions

We proposed to model cooking recipes with cooking programs. We designed a program scheme and annotated a set of programs for cooking recipes and we presented an approach for learning to predict programs from food images and recipes. Experimental results showed that projecting the common space between images and recipes into programs can improve retrieval results. Finally, we showed how we can generate food images by manipulating a program. We hope that programs will open new directions such as allowing agents to execute recipes or allowing us to extract common-sense knowledge for food from graphs.

**Limitations and societal impact.** In this work, we do not go beyond predicting the cooking procedure via programs. However, predicting nutritional value, estimating calories and their impact in our health is an important topic. Moreover, inaccurate predicted programs might not be able to be executed or lead to inedible food. Also, ingredient prediction models should be applied consciously especially in user cases with food allergies. Future work involves an analysis on the potential biases that the programs or our training data might have (e.g. towards unhealthy food dishes, towards western world with underrepresented cuisines) and the impact that this might have on the food industry.

**Acknowledgments.** This work is supported by Nestlé.
References

[17] Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Judy Hoffman, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Inferring and executing programs for visual reasoning. In ICCV, 2017. 2
Weiqing Min, Shuqiang Jiang, Shuhui Wang, Jitao Sang, and Austin Meyers, Nick Johnston, Vivek Rathod, Anoop Korattikara, Alex Gorban, Nathan Silberman, Sergio Guadarrama, George Papandreou, Jonathan Huang, and Kevin P Murphy. Im2calories: towards an automated mobile vision food diary. In ICCV, 2015. 2


Daniel Nyga and Michael Beetz. Everything robots always wanted to know about housework (but were afraid to ask). In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 243–250. IEEE, 2012. 2

Dim P Papadopoulos, Youssef Tamaaazousti, Ferda Ofli, Ingmar Weber, and Antonio Torralba. Learning cross-modal embeddings for cooking recipes and food images. In CVPR, 2017. 1, 2, 4, 5, 6, 7, 8

Alberto Sanfeliu and King-Sun Fu. A distance measure between attributed relational graphs for pattern recognition. IEEE transactions on systems, man, and cybernetics, 1983. 7


Richard Socher, Andrej Karpathy, Quoc V Le, Christopher D Manning, and Andrew Y Ng. Grounded compositional semantics for finding and describing images with sentences. Transactions of the Association for Computational Linguistics, 2014. 5


Carl Vondrick, Donald Patterson, and Deva Ramanan. Efficiently scaling up crowdsourced video annotation. IJCV, 2013. 3

Hao Wang, Doyen Sahoo, Chenghao Liu, Ee-Peng Lim, and Steven CH Hoi. Learning cross-modal embeddings with adversarial networks for cooking recipes and food images. In CVPR, 2019. 2, 5, 6, 7

Hao Wang, Doyen Sahoo, Chenghao Liu, Ke Shu, Palakorn Achananuparp, Ee-Peng Lim, and CH Steven Hoi. Cross-modal food retrieval: learning a joint embedding of food images and recipes with semantic consistency and attention mechanism. IEEE Transactions on Multimedia, 2021. 2, 5, 6

Liwei Wang, Yin Li, and Svetlana Lazebnik. Learning deep structure-preserving image-text embeddings. In CVPR, 2016. 5


